NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

SIMULATION OF A RADAR DETECTION MODEL USING THE NPS PLATFORM FOUNDATION

by

Aaron S. Ellison

March 1996

Thesis Advisor: Arnold Buss

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE SIMULATION OF A RADAR DETECTION MODEL USING THE NPS PLATFORM FOUNDATION		5. FUNDING NUMBERS
6. AUTHOR(S) Ellison, Aaron S.		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The extensive cost to thoroughly compare new radar sensor systems is a problem in today's military. Due to the shrinking defense budget, the opportunity to replace dated sensor systems, with technologically advanced systems, seldom arises. Current funding levels no longer support long term evaluations of sensor system performance. The development of new methods to measure system performance is crucial in determining the best sensor system among many alternatives. Computer simulation is one method of conducting additional trials to characterize sensor system performance. Computer simulation can aid decision makers in selecting the sensor system that best meets the needs of the current military force structure. The cost of simulation modeling is considerably less than repeated testing of the real sensor system. This research investigates the feasibility of developing a computer simulation of a radar sensor system. The scope of the research includes computer modeling of the detection process and an evaluation of model output. This simulation model is an initial step to emphasize the power of computer simulation.

14. SUBJECT TERMS Simulation Modeling, Radar Equation, Sensor Systems, Probability of Detection			15. NUMBER OF PAGES 94
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18 298-102

DTIC QUALITY INSPECTED 3

SIMULATION OF A RADAR DETECTION MODEL USING THE NPS PLATFORM FOUNDATION

Aaron S. Ellison
Lieutenant, United States Navy
B.S., United States Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degrees of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1996

Author: [illegible]
Aaron S. Ellison

Approved by: [illegible]
Arnold Buss, Thesis Advisor

[illegible]
James Eagle, Second Reader

[illegible]
Frank Petho, CAPT, USN, Chairman
Department of Operations Research

ABSTRACT

The extensive cost to thoroughly compare new radar sensor systems is a problem in today's military. Due to the shrinking defense budget, the opportunity to replace dated sensor systems, with technologically advanced systems, seldom arises. Current funding levels no longer support long term evaluations of sensor system performance. The development of new methods to measure system performance is crucial in determining the best sensor system among many alternatives. Computer simulation is one method of conducting additional trials to characterize sensor system performance. Computer simulation can aid decision makers in selecting the sensor system that best meets the needs of the current military force structure. The cost of simulation modeling is considerably less than repeated testing of the real sensor system. This research investigates the feasibility of developing a computer simulation of a radar sensor system. The scope of the research includes computer modeling of the detection process and an evaluation of model output. This simulation model is an initial step to emphasize the power of computer simulation.

THESIS DISCLAIMER

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Faster and more complex radar sensor systems are being proposed and developed. Determining the suitability of these new radar sensor systems as replacements for current systems requires extensive testing of the new system's performance. Thorough evaluation of new radar sensor systems incurs high cost and an excessive amount of time. The declining defense budget has an impact on the quality of testing performed on a given system while force reduction affects the number of resources available to conduct complete system evaluation.

The quality of a sensor system directly influences decisions made by tactical commanders. Sensor systems that provide timely and accurate information clarify a tactical commander's appraisal of events occurring in his operating environment. Proper assessment of the operating environment reduces the chance of committing crucial errors that lead to incidents such as fratricide. A sensor system that undergoes rigorous examination before entering operational service will reduce the chance of an error in judgment by a tactical commander using the sensor system.

Since limited funds reduce the extent of testing new systems, new ways of augmenting the traditional testing methods need exploration. This thesis focuses on computer simulation modeling as an affordable and practicable procedure for augmenting testing and evaluation of new radar systems. Simulation modeling of a radar system allows for unlimited runs and instant playback of test scenarios. The design of every scenario emulates the actual test environmental conditions and interactions between the radar system and its targets.

This research led to the development of a computer program that simulates a radar sensor system attempting to detect a target. Using the operating characteristics of a given radar system as the inputs, the program determines how quickly the simulated radar is able to acquire an incoming target. The program outputs the amount of time required for the simulated radar to detect the target. To compare multiple radar sensor systems, the various operating characteristics are successively fed into the program. The radar system that detects the target the earliest is the preferred radar system. The effect on radar system performance due to varying its operating characteristics is quickly and easily visualized. A graphical display of every scenario illustrates the orientation and progress of the engagement as it evolves.

This research demonstrates one possible method of augmenting traditional radar system testing. As evidenced by the results, computer simulation is capable of providing meaningful results to the user. By accurately describing the functioning of a radar system by use of computer simulation, strengths and weaknesses of the radar system are highlighted before conducting actual radar system tests. Correcting known deficiencies, as identified by computer simulation, is a more appropriate use of limited funds and resources.

ACKNOWLEDGMENTS

The author would like to express his thanks to the many people behind the scenes, whose help was instrumental in the completion of this thesis. First, is Dr. Arnold Buss, who provided the expertise in the realm of computer simulation and objected oriented programming. His guidance in the methodology of Operations Research was the key to always keeping the 'real' problem in perspective. Next, to Dr. James Eagle, for highlighting critical aspects of detection theory which if not closely monitored, will rapidly subvert any detection model. Special thanks to Dr. William Hardenburg of the Naval Research Laboratory for giving technical guidance on radar system operation. My final thanks and appreciation goes to my wife Denise, children Ryan, Lauren, and Marisa for being remarkably patient during the multitude of late nights and weekends spent by the author on his quest to finish this research.

I. INTRODUCTION

Faster and more complex radar sensor systems are being proposed and developed. These new radar sensor systems are candidates for replacing the current inventory of aging radar sensor systems. There are two main questions to address before replacing an aging radar system. First, which new radar sensor system best meets the minimum performance criterion? Second, is the potential replacement radar sensor system actually better than the existing radar sensor system?

For the tactical commander embarked in a Naval ship, a radar sensor system is the main asset for building situational awareness and making the state of the environment visible. The radar sensor system lets the tactical commander quickly locate and identify formations of friendly, enemy, and neutral shipping. Locations of friendly aircraft are discernible as well as threat aircraft locations and maneuvers. In both the surface and air environments, the tactical commander can assign friendly forces to investigate unidentified radar targets to further clarify the state of the environment. Earlier and more accurate visualization of the tactical commander's operating environment will directly affect the safe operation of friendly forces in a hostile environment.

The fundamental method of comparing alternative radar sensor systems involves conducting trials to ascertain radar sensor system performance. Attaining reliable test results requires strict adherence to conditions in the test environment. For example, to reduce variability an acceptable test environment requires the same location and weather conditions. The sea state at the time of testing is another variable that will affect test results. The cost of holding these three variables constant during testing is prohibitively high. Cost is not only incurred from installing the competing radar sensor systems on board a test vessel, but in attaining the necessary assets to act as "targets" for the radar sensor systems to detect and track.

Another disadvantage of the fundamental test method is the extensive amount of time it takes to evaluate the multiple radar sensor systems. To test an air search radar for a ship requires the test platform to transit from the pier to the designated operating area. Aircraft fly out from their home base, which can be minutes to hours away from the ship's operating area. After coordination of both units for the test scenarios, and assuming no failures of either platform or the radar sensor system that is being tested, up to an hour of time may have elapsed. If the aircraft has enough fuel to run trials for an additional hour, perhaps four good runs are achievable before returning to base for fuel and maintenance. Therefore, only a small amount of time is truly available to test a radar sensor system, and achieving a full range a performance testing is virtually impossible. Due to the high cost and limited test time available, the fundamental test method provides only a rough estimate of the true performance of the radar sensor system.

An enhancement to the fundamental method of testing is computer simulation trials of the competing systems. Computer simulation provides the following advantages:

- *Experiment control through the use of set environmental standards.*
- *The variation of test criteria in the evaluation of radar system performance.*
- *Comparison of multiple alternative radar sensor systems.*

Cost and time savings make computer simulation an attractive alternative to the fundamental method of radar system testing. [Ref. 1, p. 115]

The focus of this thesis is to develop a computerized simulation of a radar's detection process as a means to compare different radar systems' performance in a tactical environment. Of primary interest are:

- *The degree of sophistication of the model*
- *Structure of the model*
- *Required inputs to the model*

This work is thus a first step towards the use of computer simulation to evaluate different radar systems' performance under the same test and operational conditions.

This thesis starts off with a discussion of sensor systems. Next, is a discussion of the simulation development in Chapter III, followed by a mock test scenario in Chapter IV. Results of the mock test are presented in Chapter V. Conclusions and recommendations conclude this study in Chapter VI.

II. SENSOR SYSTEMS

A. BASIC SENSOR SYSTEMS

A sensor is a device that responds to actions occurring within its vicinity. A sensor system consists of a response processor that is connected to a sensor. The sensor sends signals to the response processor which will accumulate data to interpret what is happening in the vicinity of the sensor. There are many different types of sensor systems that may be applied in a variety of ways in the commercial and military world. They range from simple systems, such as photosensitive cells that will turn on a light when darkness arrives, to a complex fire control radar that will track an inbound target. This thesis will focus on complex radar sensor systems.

Electromagnetic sensors may be divided into three main types: electro-optical, laser, and radar. All sensor systems are designed to perform the same task, albeit by different methods. Their purpose is to aid the user in determining the state of the environment by detecting, classifying and tracking targets. A target is a friendly, unknown or enemy unit. Targets take the appearance of personnel for an electro-optical sensor system, armored vehicles for a laser sensor system or aircraft for a radar system. [Ref. 2, p. 1]

Regardless of the method for evaluating the state of the environment, the same basic process will be followed for all sensor systems from initial search for targets to target destruction. The first event in this process is the detection of targets. Accomplishing target detection involves searching the environment using the sensor system under manual or automatic control. Surveillance effort during the detection phase focuses on all target types in the environment.

Following the detection event comes the proper identification of targets in the environment. The objective is to correctly identify and classify the target as a friendly, unknown or enemy. Accurate identification of targets reduces the risk of fratricide. Once a target has been identified it then is placed into a specific class of weapon system. Classification of the weapon system allows the tactical commander to prepare a course of action in the event that the target turns hostile.

If the target is one of interest, the next event for the sensor system is acquisition. Difficulties in acquisition of an elusive target may arise due to target maneuvering to avoid detection or target employment of deception techniques. In this case, flexibility in changing sensor system operating parameters will increase the chance of acquiring the target of interest.

The process concludes with the target tracking event. After successful acquisition the sensor system will track the target using a recursive routine. Information related to the target such as its speed and location receives continuous updates. An accurate tracking routine predicts with precision the future location of the target from the targets past dynamics. Figure 2.1 depicts a simple flow diagram of events that every sensor system will follow.

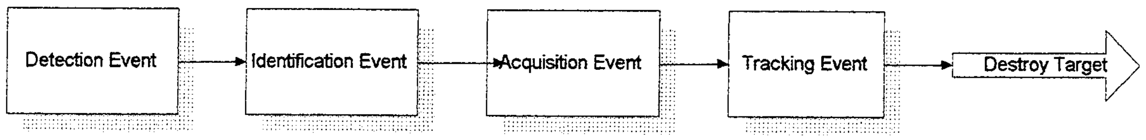


Figure 2.1 Event processing of the sensor system

B. DEFINITION OF TERMS

Common terminology describes most elements of sensor system functionality. Although the focus of this thesis is radar systems, with minor changes the terminology is applicable to other sensor systems. Figure 2.2 shows a block diagram of a simple radar system. The block diagram illustrates the basic path a received signal will follow within the radar system.

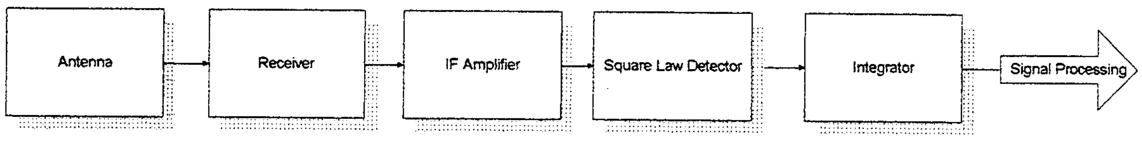


Figure 2.2 Radar sensor system block diagram. After [Ref. 3] [Ref. 2, p. 82].

During radar system operation the system transmits and receives energy pulses. The pulse duration τ is the length of time that power is transmitted during a single pulse. The peak transmitted power level P_p, pulse repetition frequency PRF of the system and τ determine the average power transmitted, given by Equation 2.1.

$$P_{ave} = P_p \tau PRF \tag{2.1}$$

Figure 2.3 depicts the calculation of the range from the sensor to the target. Time t represents the amount of time it takes for the emission and return of the transmitted energy pulse. Since the pulse propagates through the atmosphere at the speed of light c, a simplistic calculation of range is given by Equation 2.2. [Ref. 2, p. 93] [Ref. 2, p. 56]

The presence of noise in the radar receiver degrades the chance of the transmitted energy pulse, or signal, being classified as a detection. Noise comes mainly from three sources: thermal noise associated with the electronic circuits in the radar's receiver, noise from the first stage in the radar receiver, and atmospheric noise detected by the radar's

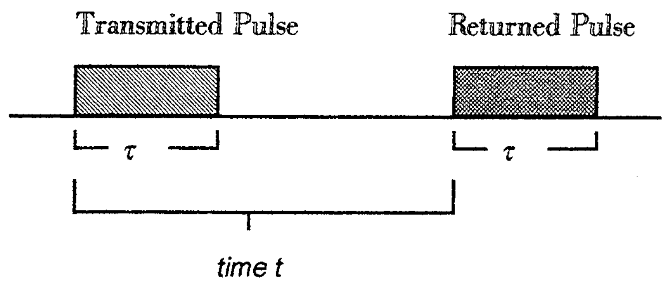


Figure 2.3 Transmitted and returned pulses. From [Ref. 2, p. 57]

$$R = \frac{ct}{2} \tag{2.2}$$

receiving antenna. Additional sources of noise come from weather conditions, terrain, and other radio frequency emitters. Upon entering the receiver, the received signal is routed to an intermediate frequency amplifier to increase the chance of a weak received signal being classified as a detection. The amplification of the return signal also causes the amplification of the noise signal. Although strengthening of the return signal through amplification is helpful for signal processing, amplification does not ensure that the minimum detectable return signal will be separable from the noise inside the radar sensor system. [Ref. 4, p. 3]

When the receiver output signal exceeds a predetermined threshold, a detection has occurred. The threshold or bias level can be exceeded by a signal consisting of noise alone. The higher the bias level setting, the less likely noise alone will exceed the bias level and trigger a detection within the radar system. If noise alone causes a detection then the false detection is referred to as a false alarm. [Ref. 4, p. 21] [Ref. 5, p. 71]

Gain is a significant parameter indicative of radar sensor system performance. Gain is an effectiveness measure regarding the ability of the radar antenna to focus its transmitted energy in a specified direction. Antenna gain G as a function of the radar systems wavelength λ and aperture area A is shown in Equation 2.3 . [Ref. 2, p. 3]

$$G = \frac{4\pi A}{\lambda^2} \tag{2.3}$$

The aperture area is the area of the radar that receives the returned power density that has been reflected from the target [Ref. 6, p. 7]. Equation 2.4 gives the aperture as a function of the antenna gain and wavelength.

$$A = \frac{G\lambda^2}{4\pi} \tag{2.4}$$

Pulse integration involves summing the voltages registered by incoming signals to determine the occurrence of target detection. The detector does not discriminate between signal plus noise voltages and noise-only voltages. Therefore, it is possible to sum only noise pulses to compare against a threshold for determination of a target detection. A signal-to-fluctuating noise ratio, known as standard deviation of noise or root mean square (RMS) noise, is compared to a threshold value. If the signal-to-fluctuating noise ratio exceeds the threshold value then a target has been detected. The advantage to pulse integration is that the summed pulses provide a better indication of the presence of a target since noise oscillates around a mean value.

If noise maintained a constant value over all returned signals, then target detection would consist simply of subtracting the noise level from the returned signal and determining if this new signal is greater than the threshold. If the signal minus constant noise is greater than the threshold, then a detection is registered. Therefore, on any given pulse an accurate assessment of the presence of a target is possible and pulse integration is not necessary.

In the case of fluctuating noise, the noise cannot be as easily extracted from the total signal since the magnitude of noise varies on each pulse. Accepting a signal-to-noise ratio greater than the threshold in this situation constitutes a false detection. Possibly, a large portion of the total signal consists of noise since noise can add enough strength to a signal to exceed the threshold setting. By integrating pulses, the true signal could be averaged over *n* pulses to attain a positive target detection. As a consequence, increasing the number of pulses integrated increases the chance of positively detecting a target at the expense of a lengthening time until detection. [Ref. 3, p. 25] [Ref. 5, p. 73]

Transmitted energy from a radar sensor system experiences many types of signal attenuation, or loss, from the time it leaves the antenna, travels through the atmosphere, and ultimately returns to the receiver. Some examples are scan distribution loss, target fluctuation loss, integration loss and atmospheric loss. System loss is an aggregate of loss due to radar system hardware. Atmospheric loss results from absorption of signal energy by the atmosphere during signal propagation. The effect of loss on radar performance is the reduction in signal-to-noise ratio. Reducing the signal to noise ratio decreases the chance of detecting a target. [Ref. 4, pp. 15-16]

C. RADAR EQUATION

The radar equation forms the basis for determination of target range. The maximum range of a radar sensor system is a function of three fundamental parameters:

transmitted power, antenna gain, and receiver sensitivity. Equation 2.5 defines the functional form of the radar equation with the signal-to-noise ratio as a function of radar system and environmental parameters. From Equation 2.5 it is obvious that the signal-to-noise ratio is inversely proportional to target range raised to the fourth power.

$$\frac{S}{N} = \frac{P_{ave} G A \sigma_{RCS} E n}{\left[4\pi R^2\right]^2 \left[\sigma^2_{noise} + \sigma^2_{clutter}\right] \left[L_{sys} L_{atm}\right]} \qquad \textbf{(2.5)}$$

where

P_{ave} = average transmitted power

A = antenna aperture

G = antenna gain

σ_{RCS} = target's radar cross section

E = integration efficiency

n = number of pulses integrated

R = range to target

σ^2_{noise} = root mean squared thermal noise power

$\sigma^2_{clutter}$ = root mean squared clutter power

L_{sys} = system losses

L_{atm} = atmospheric losses

Antenna aperture is the area of the antenna that receives the focused returned energy pulse. Integration efficiency measures how well the pulse integrator operates, where a flawless integrator has an efficiency of one. Clutter causes distortion of the reflected signal due to excess scattering of the signal energy or absorption by some other medium.

Equation 2.6 demonstrates that the signal-to-noise ratio at time of detection determines target range. [Ref. 2, pp. 89, 95] [Ref. 4, p. 28]

$$R = \left[\frac{P_{ave} G A \sigma_{RCS} E}{16\pi^2 \left[\frac{S}{N}\right] \left[\sigma^2_{noise} + \sigma^2_{clutter}\right] \left[L_{sys} L_{atm}\right]} \right]^{\frac{1}{4}} \qquad \textbf{(2.6)}$$

D. PARAMETERS

The radar equation in Equation 2.6 illustrates that average power, gain and aperture define the basis of radar sensor system operation. Although these parameters are not the only parameters that describe the functioning of a radar sensor system, they provide sufficient information to evaluate the performance characteristics of the radar sensor system. The list of sufficient parameters that determine radar range are provided in Table 1.

Calculations based on range are a function of the target that the radar sensor system attempts to detect, the performance characteristics of the radar sensor system and atmospheric conditions. The radar cross section refers to the area of the target that reflects energy back to the radar sensor system receiver. The radar cross section of a target varies dependent on the aspect angle of the target in relation to the beam of the radar. As the target moves through the atmosphere its radar cross section fluctuates in intensity. Fluctuation of the radar cross section occurs between radar sensor system interrogations of the target. Target interrogation happens on a pulse-to-pulse or scan-to-scan basis. [Ref. 4, pp. 37-38, 60]

PARAMETER	SYMBOL	UNITS
Radar Cross Section	σ_{RCS}	meters^2
Gain	G	dB
Wavelength	λ	meters
Power	P_{ave}	Watts
Aperture	A	meters^2

Table 2.1 Defining parameters for radar sensor system. After [Ref. 2, p. 25]

A target's radar cross section is generally modeled as a composite of a finite number of points that reflect transmitted radar energy back to the radar sensor system receiver. The composite radar cross section, equal to the sum of the individual cross sections, approximately follows the Rayleigh distribution [Ref. 4, p.61]. Another case of fluctuating radar cross section combines a dominate reflecting point and many lesser reflecting points. This one-dominant scatterer case of target reflectivity results in the Chi Square distribution with four degrees of freedom. The multiple scatterer and one dominant scatterer form the two main categories of fluctuating radar cross sections. [Ref. 2, p. 117] [Ref. 4, pp. 59-61]

III. SIMULATION DEVELOPMENT

A. DETERMINISTIC VS. STOCHASTIC

The cookie cutter is the basic detection model for a radar sensor system as shown in Figure 3.1. The cookie cutter model easily integrates into any radar sensor system scenario. The sensor has a detection range of radius *R* that covers the radar sensor system for 360 degrees. In the cookie cutter model the radar sensor system's location is at the center of the circle. A detection occurs when the distance between the sensor and the target is less than or equal to the radius *R*. In its simplest form, cookie cutter detection guarantees a 100 percent chance of target detection. Since the signal-to-noise ratio fluctuates from scan to scan, the chance of detection at every point within the

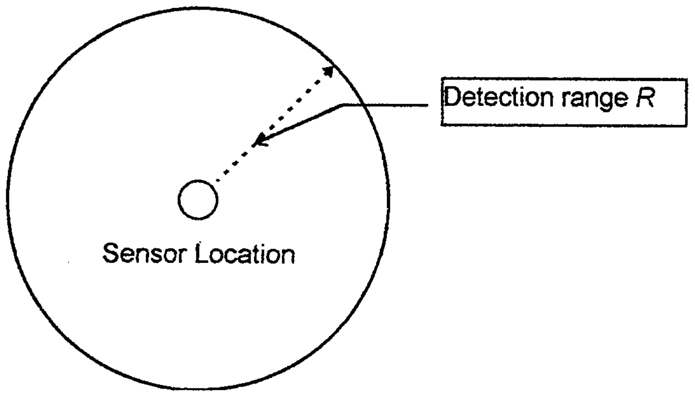


Figure 3.1 Cookie cutter detection model.

detection radius of the radar sensor system cannot be 100 percent. A comparison of radar sensor system performance using this simple detection model will not be sufficient to adequately evaluate the performance of radar sensor systems. If the range of detection is the performance measure, then the sensor with the farthest detection range will always detect the target first. Therefore, the simple cookie cutter model does not describe radar sensor system performance at the level of detail needed for the comparison of alternatives. If the simple cookie cutter model is expanded to incorporate other events that affect the detection process, it is possible to develop a model for comparing alternative sensor systems.

A stochastic detection model more accurately reflects changes that occur in the physical operating environment of the radar sensor system between successive scans. These changes include, but are not limited to, fluctuation of the target's radar cross section, noise levels in the radar sensor system receiver and noise generating atmospheric conditions. The consequence of changes in the physical environment, between radar scans, is the oscillation of the strength of the signal and noise at the receiver output. Random events in the environment affect the probability that the radar sensor system detects a target at some range $r \leq R$.

Evaluation of the stochastic detection model requires use the proper analytical tool. When combined, the radar sensor system events described in Chapter 2 may be viewed as a process, and the detection event as a sub-process, in the operation of the radar sensor system. Simulation modeling is a valuable tool for process analysis. For a radar sensor system, simulation modeling allows

- *Time based analysis of the detection process.*
- *The ability to replay detection scenarios.*
- *The capability to explicitly model randomness*
- *Analysis of factors that directly and indirectly influence the detection process*

B. PROBABILITY OF DETECTION

When pulse integration occurs after the square law detector it is called post detection integration and is depicted in Figure 3.2. Noncoherent pulse integration

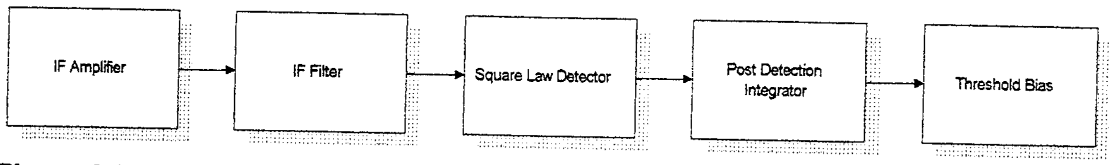


Figure 3.2 Signal processing. After [Ref. 7, p. 138]

happens when the phase relationship of the received signal as compared to the originally transmitted signal is destroyed in the intermediate frequency filter. The efficiency E of noncoherent pulse integration is a positive value less than one. Summation of the pulses during noncoherent integration provides input to the threshold bias decision circuits. A detection takes place if the input exceeds the threshold bias. [Ref. 2, pp. 81-85] [Ref. 7, pp. 137-139]

Single-hit probability of detection is the chance of detecting a target. An energy signal transmitted by a radar sensor system contains at least one pulse. This pulse or group of pulses has a finite amount of time T_d to reach the target and return to the receiver while the radar sensor system scans the environment. For T_d, the time available for detection, the relationship holds that $T_d >> 1/(\textit{Pulse Repetition Frequency})$. The radar sensor system receives a fixed number of pulses during the time interval T_d and pulse integration then takes place. Equation 3.1 defines the relationship between the number of pulses available for integration N, PRF, width of the radar beam θ_{BW} and radar scan rate γ. [Ref. 7, p. 144-145] [Ref. 4, p. 40]

$$N = \frac{\theta_{BW} PRF}{\gamma} \tag{3.1}$$

Single-hit probability of detection is a function of false alarm time. The false alarm time is the amount of time that elapses between instances of noise alone exceeding the threshold bias level. Stated another way, it is the time required for noise alone to exceed the bias level with probability of 0.5. The probability of false alarm is given in Equation 3.2 as a function of threshold bias level Y_b and N , the number of pulses integrated. I, in Equation 3.2, is the incomplete gamma function.

$$P_{fa} = 1 - I\left[\frac{Y_b}{N^{0.5}}, (N-1)\right] \qquad \textbf{(3.2)}$$

Equation 3.2 is closely approximated by Equation 3.3 below.

$$P_{fa} \approx \frac{NY_b^N \exp(-Y_b)}{N!(Y_b - N + 1)} \qquad \textbf{(3.3)}$$

Knowing the probability of false alarm and the number of pulses integrated, a threshold bias level can be found. [Ref. 5, pp. 71,77] [Ref. 4, pp. 21-23]

The Swerling models are the most commonly used models to describe the single-hit probability of detection for objects with fluctuating radar cross sections. Swerling Cases 1 and 2 describe targets whose radar cross section is a composite of a finite number of scattering points. Case 1 specifically addresses radar cross sections that fluctuate from scan-to-scan. During T_d the radar cross section fluctuations among individual pulses are highly correlated. Little to no correlation of radar cross section fluctuations is evident during the time between radar sensor system scans. This leads to independent radar cross section fluctuations for the Swerling Case 1 model. Equation 3.4 gives the probability density function for the signal-to-noise ratio for a target with a fluctuating radar cross section.

$$f(x) = \frac{1}{\bar{x}} e^{\frac{x}{\bar{x}}} \tag{3.4}$$

where

x = observed signal-to-noise ratio

$\bar{x}$ = mean signal-to-noise ratio

The Swerling Case 2 model adheres to the same concept of multiple independent scattering points, but considers pulse-to-pulse fluctuations of the target's radar cross section. The Swerling Cases 3 and 4 models address target radar cross sections with one-dominate scattering area. [Ref. 4, pp. 59-63] [Ref. 8, pp. 122-126]

The maximum detectable range R_o of a target defined by its radar cross section σ_{RCS} is found from Equation 2.6 by setting the signal equal to noise, resulting in a signal-to-noise ratio of 0dB. R_o is assumed to be the farthest range at which the radar sensor system could detect the target. Equation 3.5 defines the relationship of some range $R \leq R_o$ and the detection signal-to-noise ratio x. [Ref. 7, p. 137]

$$x = \left(\frac{R_o}{R}\right)^4 \tag{3.5}$$

To express the single-hit probability of detection P_d as a function of the threshold bias Y_b, the mean signal-to-noise ratio $\bar{x}$, and the number of pulses integrated N for a Swerling Case 1 target, the exact formulas are given in Equations 3.6 and 3.7 for $N = 1$ and $N > 1$ respectively. Due to the computational complexity of the exact single hit probability of detection in Equation 3.7, an approximation is provided in Equation 3.8 [Ref. 8, p. 127].

$$P_d = \exp\left[\frac{-Y_b}{1+\bar{x}}\right] \quad \textbf{(3.6)}$$

$$P_d = 1 - I\left[\frac{Y_b}{\sqrt{N-1}}, N-2\right] + \left(1+\frac{1}{N\bar{x}}\right)^{N-1} I\left[\frac{Y_b}{\left(1+\frac{1}{N\bar{x}}\right)\sqrt{N-1}}, N-2\right]\exp\left[\frac{-Y_b}{1+N\bar{x}}\right] \quad \textbf{(3.7)}$$

$$P_d \approx \left(1+\frac{1}{N\bar{x}}\right)^{N-1} \exp\left[\frac{-Y_b}{1+N\bar{x}}\right] \quad \textbf{(3.8)}$$

When $N\bar{x} > 1$, Equation 3.8 is simplified to Equation 3.9where P_d is a function of $\bar{x}$, N and Y_b.

$$P_d = \exp\left[\frac{(Y_b - N + 1)\bar{x}}{N}\right] \quad \textbf{(3.9)}$$

Also, Equation 3.5 and 3.9 yield Equation 3.10 as a useable formula to find the single-hit probability of detection. [Ref. 8, pp. 124-129]

$$P_d = e^{-u^4} \quad \textbf{(3.10)}$$

where

$$u = \left(\frac{R}{R_o}\right)\left(\frac{Y_b - N + 1}{N}\right)^{\frac{1}{4}}$$

C. SIMULATION DESCRIPTION

The simulation takes radar sensor system parameters as inputs and determines a random range of detection based on radar performance parameters and target radar cross section. The basis for building the simulation model is the Naval Postgraduate School Platform Foundation. The NPS Platform Foundation provides off-the-shelf building blocks for simulation experiments and includes a graphical display of the simulation experiment scenario. MODSIM II is the object oriented language used to program the modules of the NPS Platform Foundation. The structure of the NPS Platform Foundation consists of an imaginary line that divides the generic modules common to all simulations (below the line) and scenario specific modules designed by the user (above the line). Above-the-line modules utilize the basic functions of below-the-line modules. Appendix A includes a further discussion of the NPS Platform Foundation. [Ref. 9, pp. 3-5]

The simulation model focuses on the detection process of the radar sensor system, although a complete scenario from detection to target destruction can be run. The model provides statistics concerning the time until target of detection and about the target's radar cross section. An example of a scenario consists of a ship with a radar sensor system which is detecting an inbound aircraft flying through the radar sensor system's area of coverage as illustrated in Figure 3.3. The farthest range ring from the ship in Figure 3.3 is the theoretical maximum range of the radar sensor system. The next range ring indicates the maximum detectable range of the target R_o, based on the target's radar cross section and the radar's parameters. The range ring closest to the ship is the random detection range of the target. This example shows the case of one target and one radar sensor system. The simulation model can evaluate multiple radar sensor systems and multiple targets.

1. Model Assumptions

The level of detail chosen for the simulation model was designed to reflect the minimum number of parameters needed to describe the basic functioning of the radar sensor system in a generic environment. Conditions that might readily be observed in the physical environment need not be explicitly developed in the simulation model.

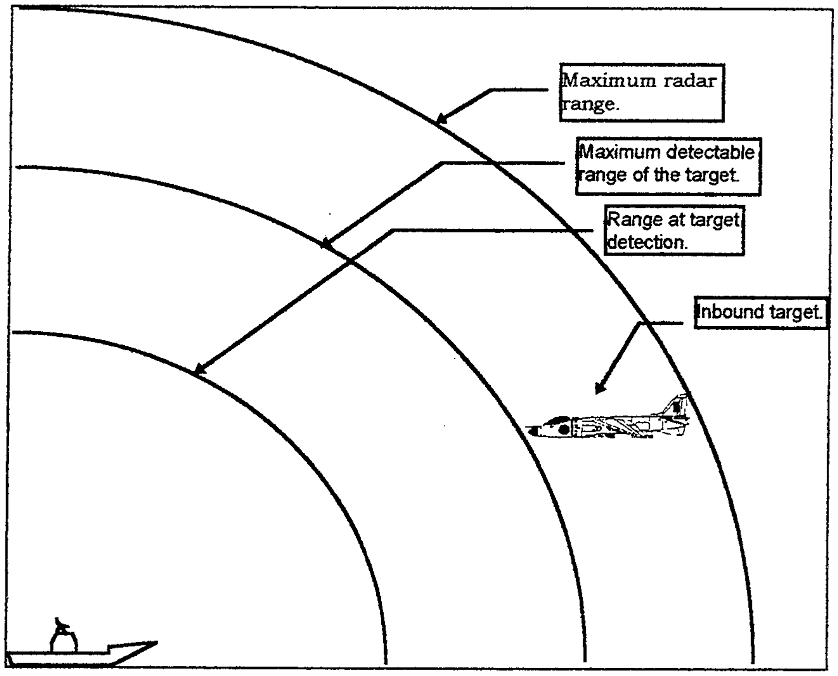

Maximum radar range.
Maximum detectable range of the target.
Range at target detection.
Inbound target.

Figure 3.3 Typical engagement scenario.

As stated previously, the single hit probability of detection is a function of the returned signal. The returned signal fluctuates between successive radar sensor system scans. Signal correlation exists at the output of the receiver, but only for a short time duration that is approximately equal to the inverse of the filter bandwidth [Ref. 4, p. 39]. This time corresponds to the amount of time the target is illuminated during a single scan. Decorrelation of returned signals, from consecutive scans of the radar, arise because the elapsed time between scans is much greater than the amount of time the signals remain correlated. The resulting decorrelation allows each scan to be considered a discrete independent look at a target. Since this model focuses on the initial detection of the target, not repeated detections, signal independence simplifies and expedites mathematical calculations. The penalties for assuming signal correlation between scans are a need for excessive computer memory and longer simulation run time.

The distance from the target's initial location to the sensor is set at some value greater than R_o and the target always proceeds towards the sensor. The occurrence of the initial detection is crucial for modeling the detection phase of the radar since it is the mechanism that determines the response of the radar sensor system and influences the decisions made by the tactical commander. The probability of not detecting a target, P_{nd}, on a single discrete scan is given by Equation 3.11. Since the radar sensor system will scan a target more than one time, the expression in Equation 3.12 shows that the probability of not detecting a target by scan i is the product of the probability of not detecting the target on all previous scans. Equation 3.13 shows the probability of detecting the target by scan i. [Ref. 4, p. 39] [Ref. 7, pp. 58-60, 159]

$$P_{nd} = 1 - P_d \tag{3.11}$$

$$P_{nd}(i) = \prod_{k=1}^{i} \left(1 - P_d(k)\right) \tag{3.12}$$

$$P_d(i) = 1 - \prod_{k=1}^{i} \left(1 - P_d(k)\right) \tag{3.13}$$

The initial location of a target is assumed to be outside of the range R_o, assuming that the coverage area of the radar sensor system is in the shape of a hemisphere. A target approaching a radar sensor system originates from some point A, normally a considerable distance from B, as it proceeds to point B. In this tactical engagement simulation model, point B will always correspond to a specific radar sensor system's location.

2. Model Inputs

Model inputs depend on the type of analysis to be performed. The NPS Platform Foundation has its own set of requisite model inputs. Table 3.1 summarizes the additional inputs for running this radar sensor system detection model. To characterize the platform that the radar sensor system desires to detect requires an input of the radar cross section of the potential target. Describing the radar sensor system requires ten additional inputs. These attributes are: average power P_{ave}, aperture size A, integration efficiency E, number of pulses integrated n, wavelength λ, false alarm number n', filter bandwidth B, and receiver temperature K°, system loss L_{sys}, and the time delay between successive scans of the same point in space T_S. The clutter power $\sigma^2_{clutter}$ and atmospheric loss L_{sys} explain the state of the operating environment of the radar sensor system.

Of the attributes listed in Table 3.1, the scan time is the only parameter not represented in the radar equation given in Equation 2.5. The scan time sets the proper time interval for the independent scans of the target. The parameters B and K° relate to the thermal noise power in the receiver and is shown in Equation 3.14 [Ref. 6, p. 133]. As evidenced in Equation 2.5, thermal noise plays an important role in the radar equation.

D. IMPLEMENTATION

The modules developed for this thesis were specifically designed to evaluate a radar sensor system's detection phase. The additions to the NPS Platform Foundation include both above-the-line and below-the-line modifications. Below-the-line changes consisted of ensuring the correct parameters were passed to the new modules such that the integrity of the NPS Platform Foundation remained intact. Other changes were made to add the additional parameters needed to model the radar sensor system as shown in Table 3.1. Appendix B provides the source code of the below-the-line changes. The above the line changes were made to develope the essence of the radar sensor system operating in a tactical environment. The structure is built around an environmental object as shown in Figure 3.4. The environment transfers information between the target and the sensor. The sensor is an independent entity, but is linked to its host platform through a virtual sensor. Therefore, referencing the sensor or the host platform leads to the same effect.

Name	**Symbol**	**Units**
Radar Cross Section	σ_{RCS}	Meters^2
Average Power	P_{ave}	Watts
Integration Efficiency	E	------
Pulses Integrated	n	------
Wavelength	λ	Meters
False Alarm Number	n'	------
Filter Bandwidth	B	Hz
Receiver Temperature	T	K°
Scan Time	T_S	Seconds
Clutter Power	$\sigma^2_{clutter}$	Watts
System Loss	L_{sys}	------
Atmospheric Loss	L_{atm}	------

Table 3.1 Radar attribute inputs for the simulation model.

$$\sigma^2_{noise} = kTB \quad \textbf{(3.14)}$$

where

$k = 1.38 \times 10^{-23} \frac{\text{watts}\cdot\text{sec}}{\text{deg}}$

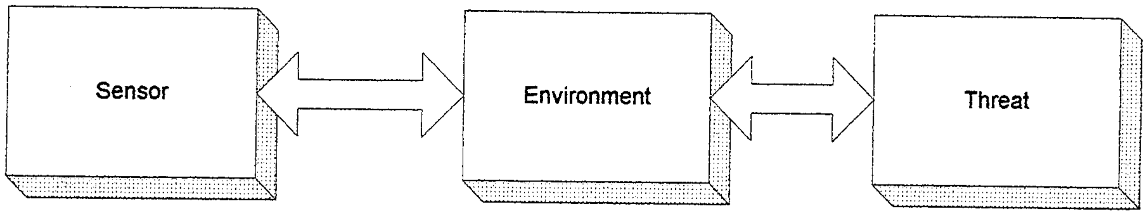


Figure 3.4 Two-way information flow between environment, sensor, and threat.

1. Environment

The physical atmosphere through which the signal travels is simulated by a variable called the environment as found in Appendix B. The environment is the source for determining when and where a detection takes place. The environment knows the performance characteristics of the radar sensor system, speed and three dimensional location coordinates of the platform and target, and the target's radar cross section. The role of the environment is to randomly select a detection time from the cumulative distribution function (CDF) for the probability of detecting the target. Construction of the CDF is based on Equation 3.13, where each scan i is determined by the radar sensor system scan time T_S. Every T_S seconds the radar will have a discrete and probabilistically independent look at the target with a corresponding probability of detection.

The process for building the CDF requires determining the location coordinates of R_o based on the movements of the platform and the target. Next the environment finds the ranges and location coordinates of the target for each discrete radar sensor system scan. Since this thesis focuses on radar detection, the ranges and location coordinates are found only while the target travels inbound to the sensor. The cumulative probability of detection is computed by an iterative process of summing the probability of detection for all prior radar sensor system scans. At this point, for each radar scan, the environment has computed a range where the scan transpires, the probability of detection on the scan, and the cumulative probability of detection on the scan.

The accumulated probabilities form a discrete step function from which a random scan representing the detection is drawn. The method for drawing the random scan comes from the inverse transform method of generating discrete random variates [Ref. 1, pp. 469-474]. The time associated with the scan represents the additional time until detection of the target after reaching the range R_o. To clarify, R_o represents the farthest range that a radar sensor system, based on its operating characteristics, can detect a target with a radar cross section of size σ_{RCS}. Calculating the time for the target to proceed form its initial location until intercepting a hemisphere around the sensor of radius R_o is always a fixed time of travel. Therefore, since the target cannot be detected at any range greater than R_o, the only randomness comes in to play after the target reaches its maximum detectable range. The random drawn time is added to the time the target intercepts R_o as determined by the NPS Platform Foundation. When the simulation time equals the sum of the two times a detection is registered. If the graphics screen is utilized to view the scenario, the user will notice the range ring of the sensor changes from green to red and the status ring of the target changes from yellow to blue at the instant the target is detected.

2. Radar and Threat

The first step for initializing the radar sensor system detection process is determining the range R_o from the radar equation. This range is where the signal to noise ratio equals one. From Equation 3.5 the relationship between range and signal to noise ratio has been established. The radar object then determines Y_b, the threshold setting.

The threshold is automatically set by a recursive routine which can be found in Appendix B [Ref. 4, p. 486]. At the time of detection the radar object gathers statistics related to the signal to noise ratio that are used for analysis. The threat object is merely an interface to provide information to the radar via the environment with out divulging hidden information concerning the target's location, speed, and radar cross section.

IV. PROOF OF CONCEPT

The best method of demonstrating the concept of using simulation as an aid to radar sensor system analysis is through a scenario emulating actual events. The scenario compares the performance of two competing radar sensor systems. The expected time until detection is the measure of effectiveness for the scenario. After determining the expected time of detection of each system against a target of a given radar cross section, the combined signal and noise distribution is examined. The examination focuses on the observed combined signal at the time of target detection. From this analysis conclusions are drawn to describe the functionality and compatibility of the radar sensor system.

A. MOCK SCENARIO

The scenario involves the evaluation of the SPS-30 and SPS-49 air surveillance radars. The SPS-30 and SPS-49 are in competition to become the replacement for the current air surveillance radar system, which has reached the end of its service life. The manufacturers view the chance to produce the follow-on to the current air surveillance radar sensor system as an extremely profitable government contract. Although the manufacturers both claim to meet the military specification for the air surveillance radar sensor system design, only one will be chosen as the follow on to the current air surveillance radar sensor system.

The high cost of installing the competing radar sensor systems onboard a naval vessel for evaluation and limited availability of test assets has led to the use of simulation modeling for initial exploratory analysis of the radar sensor systems performance. After gaining knowledge of the capabilities and weaknesses of the radar sensor system through simulation output analysis, resources can then be allocated to investigate and test specific areas of air surveillance system performance to acquire more precise data for comparing the two systems.

The scenario has the SPS-30 and SPS-49 test being conducted in an controlled ocean environment. The test involves the detection of a target with a radar cross section size of 5 square meters. The target flight profile involves a constant altitude of 500 feet and constant airspeed of 500 knots, commencing from a distance of 100 nautical miles from the ship. The metric for the test is the expected time of detection, calculated from the time the target has commenced its inbound run. The smaller the expected time of detection, the better the performance of the air surveillance system under this measurement criteria. Operational characteristics of the two radar sensor systems is fictitiously provided by the radar sensor system manufacturers. Table 4.1 lists the input parameters to the simulation model that specify the radar sensor system's operating characteristics.

Parameter	**SPS-30**	**SPS-49**
P_{ave}	9000	13000
A	8.82	6.24
E	0.98	0.95
n	5	7
λ	0.15	0.319
n'	100000	100000
T_S	6.0	5.0
T	1200	800
B	2000000	2000000

Table 4.1 Scenario inputs. After [Ref. 10, pp. 222, 224]

The actual events in a scenario are illustrated in Figure 4.1 through Figure 4.7. When viewing the scenario on a computer monitor specific colors describe significant events. Figure 4.1 displays the initial setup of the ship and the target. The small status ring around each platform indicates there has not been a detection. Visually on a computer monitor the initial color of the status ring is yellow. In Figure 4.2, the radar's range ring representing the maximum detectable range of the target is added to the ship. When viewing the scenario on a computer monitor the initial color of the radar range ring is green. After the addition of the radar range ring the simulation begins. The target proceeds inbound to the ship in Figure 4.3, as the ship proceeds on its course. Although the target has crossed the maximum detectable range, Figure 4.4, its status ring shows that it has not been detected by the radar sensor system. The status ring, when graphically displayed on a monitor, retains its yellow color. Figure 4.5 depicts the detection event. Viewed on a monitor, both the status ring of the target and the radar sensor range ring simultaneously change color. Maroon identifies the target in a detected status and the ship's range ring changes to red indicating the presence of a target. As the target continues inbound, Figure 4.6, its status ring continues to display a detected status. The simulation concludes when the target overflies the ship as pictured in Figure 4.7.

B. SCENARIO ASSUMPTIONS

The modeling assumptions for this scenario, and the simulation model in general, are devised to follow common reason and decrease computer programming complexity. The first assumption for any scenario involving this simulation model is that the initial location of the incoming target is greater than R_o. Also, target's mission is to attack the ship platform. Therefore, the target will proceed to the last known coordinate position of the ship. The target's radar cross section fluctuates in accordance with the Swerling Case 1 target model. Provisions have been made to easily add other Swerling models to

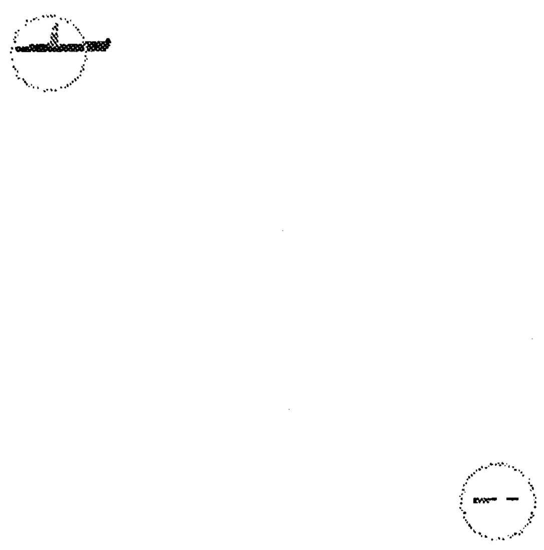


Figure 4.1 Initial setup of the ship and the target.

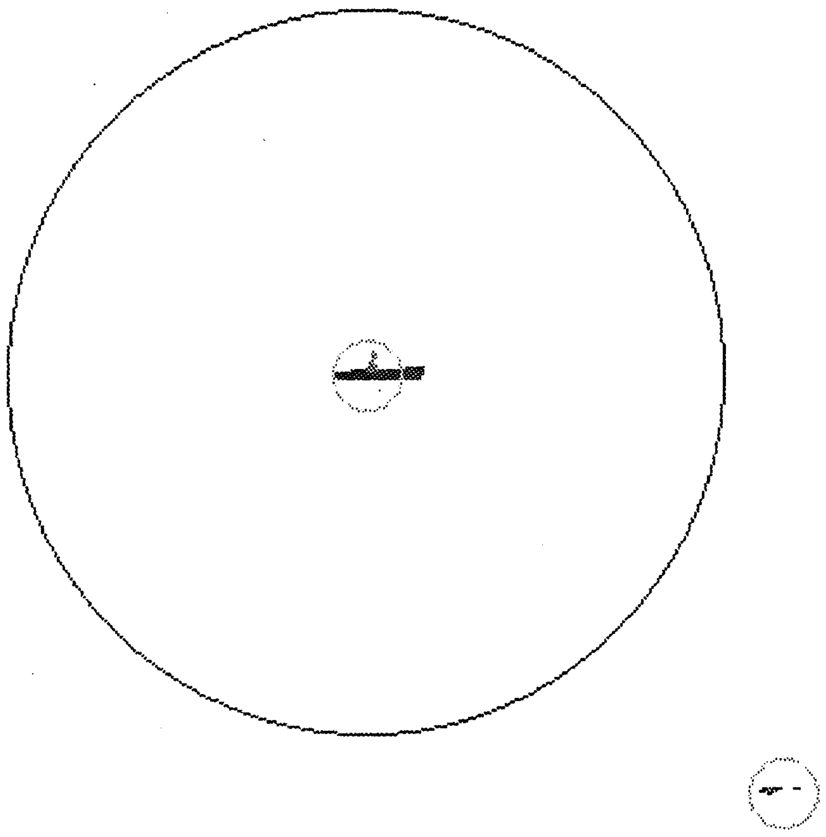


Figure 4.2 Sensor range ring established.

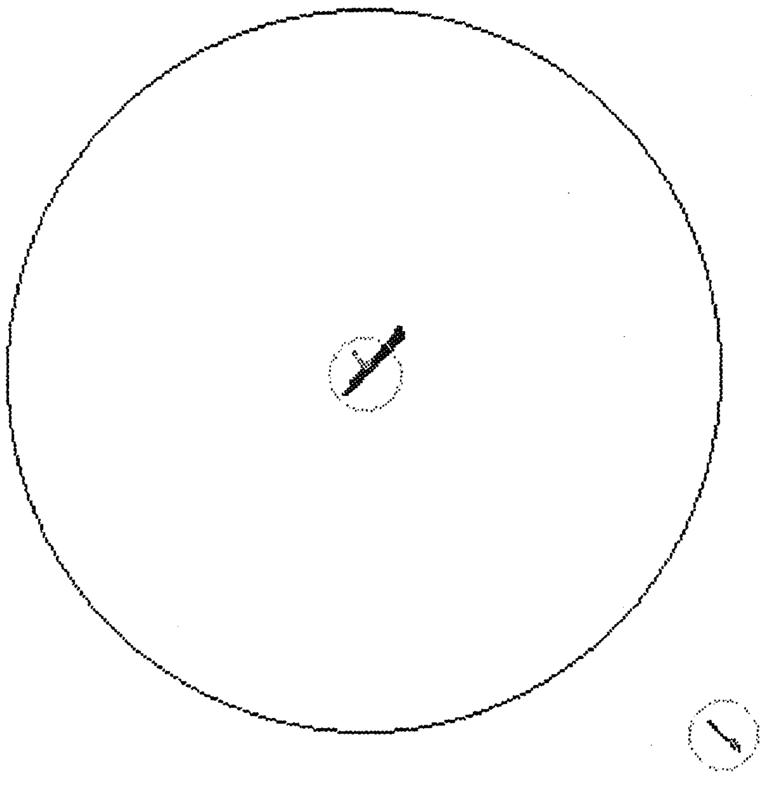


Figure 4.3 Ship steady on course and target inbound.

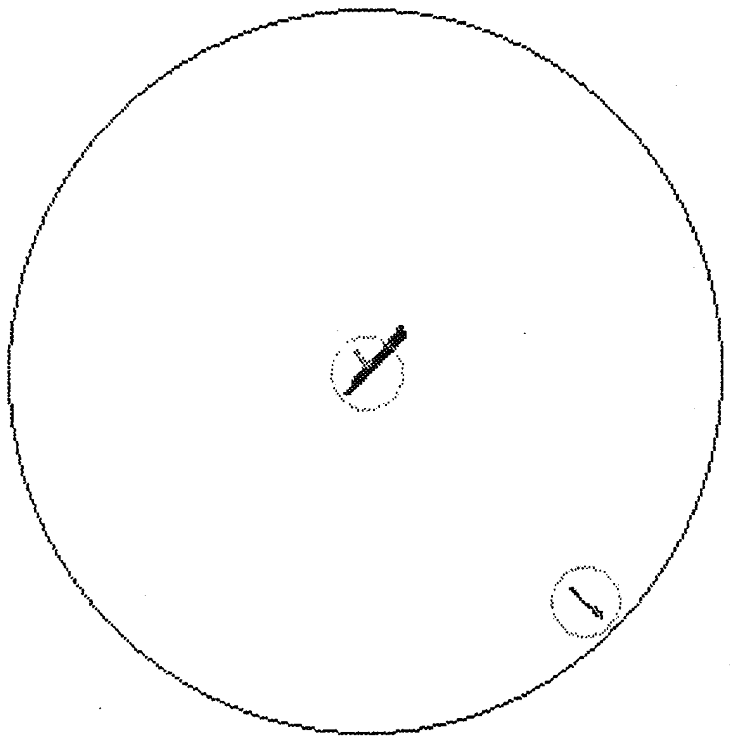


Figure 4.4 Target crossing maximum detectable range.

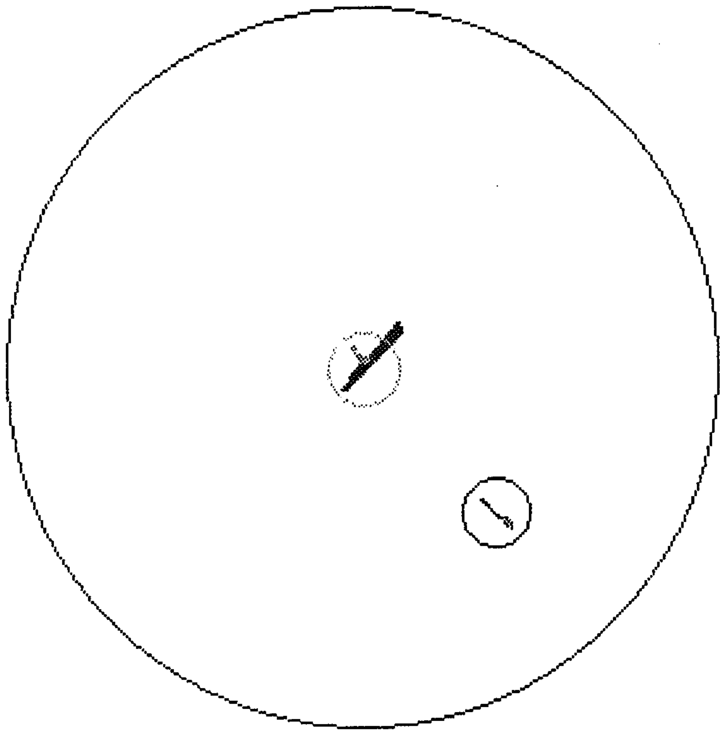


Figure 4.5 Radar detects target.

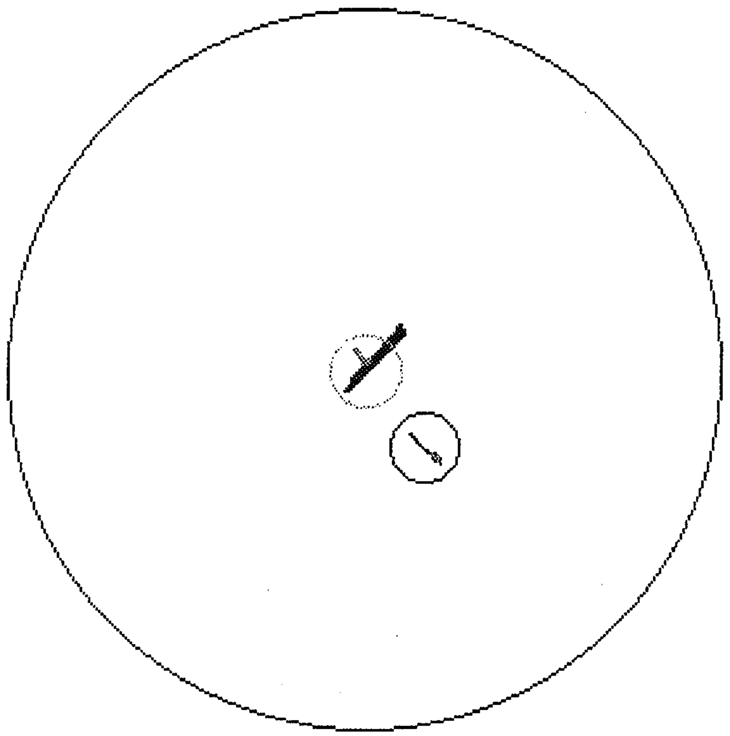


Figure 4.6 Detected target proceeding inbound.

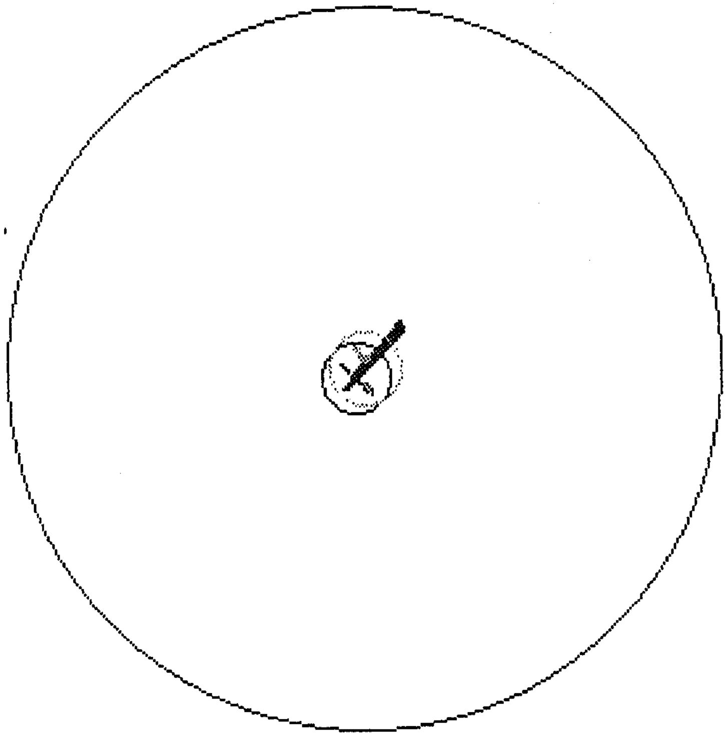


Figure 4.7 Simulated engagement complete.

the current computer code. The last general assumption stipulates that all range calculation are given in slant range to the target.

Assumptions in this scenario assert that the received signal does not experience the corrupting effect of background clutter. The parameters integration efficiency, pulses integrated, false alarm number, receiver temperature, and filter bandwidth are not true operating values of the two radar sensor systems. These values merely portray possible actual values. The simulation run is complete when the target passes overhead the ship. Because this scenario evaluates air surveillance radar sensor systems in a tactical environment, the speed of the incoming target greater than the speed of the ship.

V. RESULTS

The expected time until target detection was measured for the SPS-30 and the SPS-40 radar sensor systems based on the input parameters from Table 4.3 The procedure for collecting the data involved taking data samples of the elapsed time derived from the time the target commenced its inbound run until it was detected. The samples from repeated runs of the scenario were then combined to calculate an estimate of the mean time until detection. The point estimate is then bounded by confidence intervals to further clarify the accuracy of the estimate.

The simulation experiment is replicated until a reasonable estimate of the mean time until detection is found. The stopping rule for the simulation model is the relative error of the mean time to detection. The relative error γ specifies the percent error in the estimate of the mean $\overline{X}$ from the true value of the mean μ. Equation 5.1 shows the relationship between the mean values and the relative error [Ref. 1, p. 537]. Based on the

$$\gamma = \frac{|\overline{X} - \mu|}{|\mu|} \qquad \textbf{(5.1)}$$

magnitude of the relative error, the number of replicates the simulation performs is related to the half-width δ of the confidence interval for the estimate of the mean. The confidence interval changes as a function of the number of replicates n. Each replicate checks the size of the confidence interval half-width and compares the half-width to the relative error as given in Equation 5.2. [Ref. 1, pp. 538-539]

$$\frac{\delta(n)}{|\overline{X}(n)|} \leq \frac{\gamma}{1+\gamma} \qquad \textbf{(5.2)}$$

For the mock scenario the relative error was chosen to be 1 percent. Each radar sensor system replicate took less than one second to run. The run time for a particular radar sensor system was less than four minutes. Based on the mean time of detection and the given scenario, the SPS-30 performed better than the SPS-49. Table 5.1 summarizes the output results.

Measure	SPS-30	SPS-49
Mean Time (minutes)	4.9192	6.4556
Confidence Interval (CI)	(4.8735,4.9649)	(6.4121,6.4992)
CI Precision	0.01	0.01
CI Width	0.0913	0.0871
Sample Variance	0.2882	0.1317
Runs	531	267

Table 5.1 Time until detection summary statistics.

The flexibility of simulation modeling is further demonstrated by extending the analysis to exploring the returned signal and noise. Noise is modeled by assuming that it comes from the Rayleigh distribution shown in Equation 5.3. The signal comes from an unknown distribution, but is a function of the fluctuating radar cross section of the target and the range of the target at the time of detection. The distribution of signal plus noise theoretically comes from the Rice distribution given in Equation 5.4. Noise signals are generated by the inverse transform method for the Weibull distribution since the Weibull(2, β) is equivalent to the Rayleigh(β), where β equals the square root of two times the total noise power σ_r^2 [Ref. 1, pp. 333-334, 490]. [Ref. 3]

$$P(r) = \frac{r}{\sigma_r^2} \exp^{\frac{-r^2}{2\sigma_r^2}} \qquad (5.3)$$

where

$$\sigma_r^2 = \sigma_{clutter}^2 + \sigma_{noise}^2$$

$$P(r) = \left[\frac{r}{\sigma_r^2} e^{-\left(\frac{r^2+A^2}{2\sigma_r^2}\right)} \right] I_0\left(\frac{rA}{\sigma_r^2}\right) \tag{5.4}$$

where

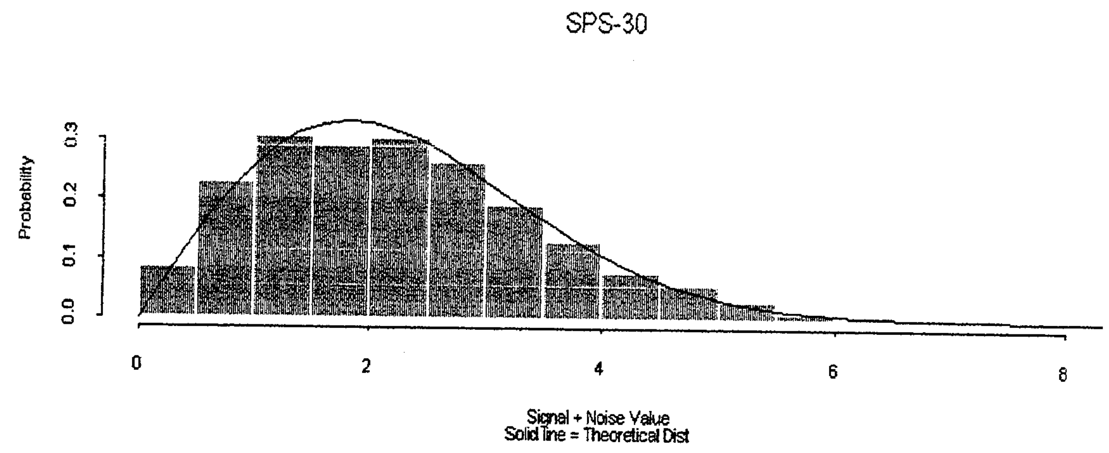
I_0 = Bessel function of order 0

A = half wave of the sinusoidal signal

The accumulation of signal plus noise data follows the same method and stopping rule as used in collecting detection time data. The received signal and fluctuating noise are computed then summed to form the composite signal. Once the composite signal data points are gathered, the observations are compared to the theoretical distribution to appraise how closely they reflect the theoretical distribution of signal plus noise. Table 5.2 consolidates the results from the two runs while Figure 5.1 plots a histogram of the observed data against the theoretical distribution. All observed data values have been rescaled by a factor of 10^{-7}.

Measure	**SPS-30**	**SPS-49**
Mean Value	8.8383	5.4060
Confidence Interval (CI)	(8.8529,9.1472)	(5.1899,5.6220)
CI Precision	0.035	0.04
CI Width	0.6179	0.4321
Sample Variance	42.5358	18.6033
Runs	1712	1531

Table 5.2 Signal plus noise summary statistics.



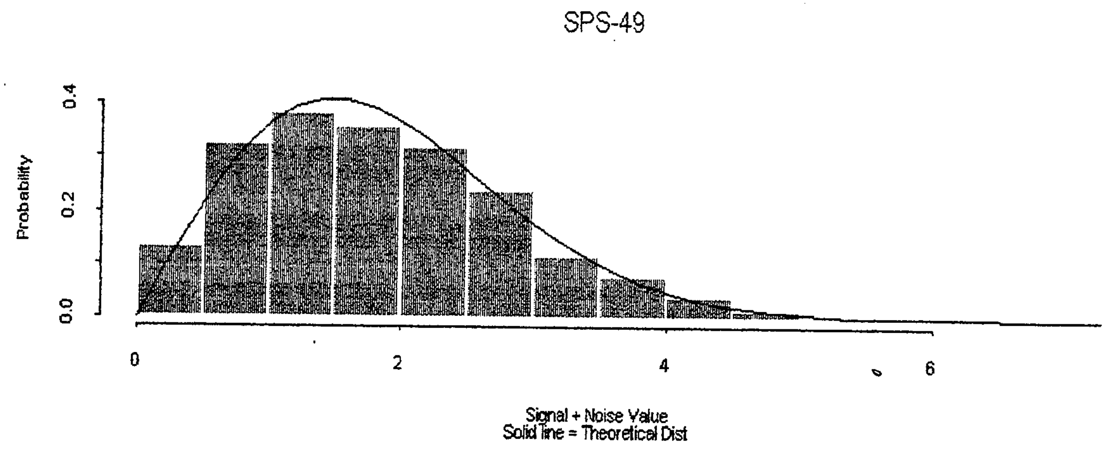


Figure 5.1 Comparison of the observed data to the theoretical distribution.

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis demonstrates the potential of simulation modeling as a tool to assist in the testing of radar sensor system. The results clearly state the benefit of the ability to replay a given detection scenario, as evidenced through the precision of the mean value estimates for the time until detection and the signal plus noise value. Additionally, the simulation shows that many replications of a scenario are needed to determine a good point estimate of the true mean of the variable under consideration. The fundamental method of comparing alternatives does not allow a vast number of trials due to the high cost of repeating the trials. Fundamental testing exercises very limited control over the environment, thereby increasing variability in the observed results.

Simulation modeling allows for exploratory analysis of sensor system processes which can lead to the discovery of new measures of sensor system performance. For example, assume that noise follows a logistic distribution rather then a Rayleigh distribution. The effect on the signal plus noise or signal to noise ratio may then be examined. Converse to studying sensor systems, one might desire to understand how a new weapon system performs against a particular enemy sensor system. By entering the basic parameters of the enemy sensor system, knowledge of the detectability of a proposed friendly weapon systems is gained.

This research evaluated a single scenario to demonstrate the feasibility of modeling radar sensor system. The ability exist to run a multitude of scenarios to investigate possible weaknesses in radar sensor system performance. Furthering of research in the area of sensor systems can be done by more accurately defining the effects of the atmosphere on the propagating signal, such as ducting and multipath. Power intensity of a reflected signal could be adapted to reflect the aspect angle of the target relative to the direction of the transmitted signal. After the desired level of detail is reached, as defined by the user, the other processes of the sensor system should be modeled.

APPENDIX A. NPS PLATFORM FOUNDATION

This appendix provides a brief overview of the NPS Platform Foundation. The Platform consists of modules designed to provide an off the shelf generic simulation package to run graphical military engagement scenarios with an emphasis towards sensor systems. The Platform Foundation is written in the object oriented language *MODSIM II* and the graphics are programmed in *SIMGRAPH II*. Scenarios are built using data input files to specify the types of systems employed, locations of the systems, and system performance characteristics and movements. The graphical map display depicts the systems and their movements on a 100x100 nautical mile grid.

The basic components of any scenario are the platforms and the sensors. The platforms can be of a specified type such as a destroyer, frigate, attack jet, or fighter jet. The platform is then assigned an identification name that relates it to a specific type of platform, for example, a platform type 'frigate' is created and a platform is given the identification name of OHPerry. The identification name OHPerry relates the platform to a platform type 'frigate'. This building convention allows for multiple forces of the same type to be present during an engagement, which is comparable to real military battle force structure. Sensors are an attribute of platform types. There is no direct link between a sensor and a platform except through the graphical movements of the platform. Whereas onboard a real naval vessel a fire control radar is physically part of the ship, in the virtual world the only connection between the fire control radar and the ship is a graphic image of the ship and a range ring representing the fire control radar scan range.

Sensor are linked to targets by a virtual sensor. Virtual sensing is the medium through which information is transferred between target and sensor. The virtual sensor maintains a queue of all targets it is tracking and calculates the time of detection based upon movements of the sensor/platform and the target. Virtual sensing keeps a platforms movements and attributes concealed from other platforms. Virtual sensing is the main force driving any scenario involving sensing.

APPENDIX B. SIMULATION CODE

The following code is a combination of new modules and critical additions to existing NPS Platform Modules that needed changes for running the simulation model. All programming codes is done in *MODSIM II* while the program is run on a UNIX workstation [Ref. 11]. The first changes to existing code was done to the modules *DetRng* and *PList*. These changes added the additional parameters needed to model the radar sensor system and target attributes. The change to PList.mod was a single line adding the radar cross section as an attribute.

Definition module DDetRng.mod:

```
FROM GM IMPORT RealmType;
FROM ListMod IMPORT QueueList;
FROM RecIO IMPORT RecIOHandleObj;

TYPE

AttributesRecType = RECORD
     AveragePower      : REAL;
     Aperture          : REAL;
     Efficiency        : REAL;
     PulseInt          : INTEGER;
     WaveLength        : REAL;
     FalseAlarmNumber  : REAL;
     ScanTime          : REAL;
     ReceiverTemp      : REAL;
     FilterBW          : REAL;
END RECORD;

SensorInfoRecType = RECORD
     SensorTypeName   : STRING;
     Realm            : RealmType;
     DefaultRange     : REAL;
     TypeSensor       : STRING; {i.e. RADAR, OPTICAL, etc.}

     Attributes       : AttributesRecType;
END RECORD;

SensorIOHandleObj = OBJECT(RecIOHandleObj[ANYREC: SensorInfoRecType])
                    END OBJECT;
```

```
 SensorMasterListObj = OBJECT(QueueList[ANYREC: SensorInfoRecType])
                               ASK METHOD GiveSensor(IN Name : STRING): SensorInfoRecType;
                             END OBJECT;

 CoupledRangeRecType = RECORD
        TgtType      : STRING;
        SensorType : STRING;
        Range         : REAL;
 END RECORD;

DetectionRangePairObj = OBJECT(QueueList[ANYREC: CoupledRangeRecType])
                               ASK METHOD ReadRangeRecs;
                               ASK METHOD CoupledRange(IN TgtType : STRING;
                               IN SensorTypeName : STRING): REAL;
                             END OBJECT;

VAR
    MasterSensorList         : SensorMasterListObj;
    DetectionRangeOracle : DetectionRangePairObj;
END MODULE.
```

Implementation module DDetRng.mod: (Changes only)

```
ASK METHOD CoupledRange(IN TgtTypeID : STRING;
                                  IN SensorTypeName : STRING): REAL;
VAR
    CurrentSensor : SensorInfoRecType;
    BEGIN
       CurrentSensor := ASK MasterSensorList TO GiveSensor(SensorTypeName);
       IF CurrentSensor.TypeSensor = "RADAR"
          RETURN(ASK Radar TO FindRo(TgtTypeID, SensorTypeName));
       ELSE
          RETURN(-1.0);
       END IF;
    END METHOD;

ASK METHOD ReadRangeRecs;
VAR
    SHRec : SHRecType;
    SHArray : SHArrayType;
    i, j : INTEGER;
    SensorRec : SensorInfoRecType;
    CharArray : ARRAY INTEGER OF CHAR;
    Rec : CoupledRangeRecType;
    MasterRangeIOHandler : SensorIOHandleObj;
    BEGIN
       NEW(MasterSensorList);
       NEW(MasterRangeIOHandler);
       ASK MasterRangeIOHandler TO ReadRecs("sensor.dat");
       ASK MasterRangeIOHandler TO FindSHRec("SensorTypes", SHRec);
       FOR i := 1 TO HIGH(SHRec.OwnedString) BY 13
          NEW(SensorRec);
          SensorRec.SensorTypeName := SHRec.OwnedString[i];
```

```
            SensorRec.Realm := ConvertToRealmType(SHRec.OwnedString[i+1]);
            SensorRec.DefaultRange := STRTOREAL(SHRec.OwnedString[i+2]);
            SensorRec.TypeSensor := SHRec.OwnedString[i+3];
            NEW(SensorRec.Attributes);
            SensorRec.Attributes.AveragePower := STRTOREAL(SHRec.OwnedString[i+4]);
            SensorRec.Attributes.Aperture := STRTOREAL(SHRec.OwnedString[i+5]);
            SensorRec.Attributes.Efficiency := STRTOREAL(SHRec.OwnedString[i+6]);
            SensorRec.Attributes.PulseInt := STRTOINT(SHRec.OwnedString[i+7]);
            SensorRec.Attributes.WaveLength := STRTOREAL(SHRec.OwnedString[i+8]);
            SensorRec.Attributes.FalseAlarmNumber :=  STRTOREAL(SHRec.OwnedString[i+9]);
            SensorRec.Attributes.ScanTime :=  STRTOREAL(SHRec.OwnedString[i+10])/60.0;
            SensorRec.Attributes.ReceiverTemp := STRTOREAL(SHRec.OwnedString[i+11]);
            SensorRec.Attributes.FilterBW := STRTOREAL(SHRec.OwnedString[i+12]);
            WriteLine("  ");
            WriteLine("Sensor " + SensorRec.SensorTypeName + " has realm " +
            RealmToStr(SensorRec.Realm) + " and default range " +
            REALTOSTR(SensorRec.DefaultRange)+ " This sensor is a "+SensorRec.TypeSensor);
            WriteLine("Attributes are Avg Power: " + REALTOSTR(SensorRec.Attributes.AveragePower));
            WriteLine("                Apeture: " + REALTOSTR(SensorRec.Attributes.Aperture));
            WriteLine("             Efficiency: " + REALTOSTR(SensorRec.Attributes.Efficiency));
            WriteLine("      Pulses Integrated: " + INTTOSTR(SensorRec.Attributes.PulseInt));
            WriteLine("             Wavelength: " + REALTOSTR(SensorRec.Attributes.WaveLength));
            WriteLine("     False Alarm Number: "
                                  + REALTOSTR(SensorRec.Attributes.FalseAlarmNumber));
            WriteLine("       Radar Scan Time: " + REALTOSTR(SensorRec.Attributes.ScanTime));
            WriteLine("       Radar ReceiverTemp: "
                                      + REALTOSTR(SensorRec.Attributes.ReceiverTemp));
            WriteLine("       Radar FilterBW: " + REALTOSTR(SensorRec.Attributes.FilterBW));
          ASK MasterSensorList TO Add(SensorRec);
      END FOR;
  END METHOD;
```

The modules *DAARON1.mod* and *IAARON1.mod* comprise the environment, radar, and threat objects. The radar determines its threshold setting in this module and the environment makes the link to create the cumulative distribution function in *IAARONCDF.mod.*

Definition module DAARON1.mod:

```
DEFINITION MODULE AARON1;

FROM GM           IMPORT LocationRecType;
FROM Plat         IMPORT PlatformObj;
FROM DetRng       IMPORT SensorInfoRecType;
FROM RandMod      IMPORT RandomObj;
FROM MathMod      IMPORT POWER;
FROM SimpleStats  IMPORT StatObj;
FROM RandMod      IMPORT RandomObj;
```

```
TYPE
  ThreatObj = OBJECT
                    ASK METHOD ObjInit;
                    ASK METHOD CalculateRCS;
                    ASK METHOD GiveMeanRCS(IN ThisThreat : STRING): REAL;
                 END OBJECT;

 EnvironmentObj = OBJECT
                       RandTime      : REAL;
                       ClutterVar    : REAL;
                       SystemLoss,
                       AtmosLoss     : REAL;
                       ActiveThreat,
                       ActivePlat    : PlatformObj;
                       FutureLocale  : LocationRecType;
                       MyRand        : RandomObj;

                       ASK METHOD ObjInit;
                       ASK METHOD GiveRCS(IN NewThreat : STRING): REAL;
                       ASK METHOD Clutter() : REAL;
                       ASK METHOD SysLoss() : REAL;
                       ASK METHOD AtmLoss() : REAL;
                       ASK METHOD SetCurrentPlats(IN P : PlatformObj; IN T : PlatformObj);
                       ASK METHOD SetEntryLocation(IN Locale : LocationRecType);
                       ASK METHOD FindDetectTime(IN PulseInt : INTEGER;IN ThisPlat : STRING;
                                                 IN ThisSensor : STRING;
                                                 IN ThisTarget : PlatformObj; IN Ro : REAL);
                    ASK METHOD PassDetectionRCS(IN ThisThreat : PlatformObj) : REAL;
                  END OBJECT;

RadarObj = OBJECT
               Yb             : REAL;
               ReceiverTemp   : REAL;
               Rmax : ARRAY INTEGER OF REAL; {MAX RANGE FOR EACH PLATFORM TYPE}
               SNRatio        : ARRAY INTEGER OF REAL;
               RCSStatKeeper : StatObj;
               RandVar1       : RandomObj;

               ASK METHOD ObjInit;
               ASK METHOD Gain(IN ThisSensor : SensorInfoRecType) : REAL;
               ASK METHOD FindRo(IN TgtType : STRING;IN SensorTypeName : STRING) : REAL;
               ASK METHOD ThermalVar(IN Csensor : SensorInfoRecType) : REAL;
               ASK METHOD Interference(IN CSensor : SensorInfoRecType) : REAL;
               ASK METHOD FindBiasLevel(IN b : INTEGER; IN NPrime : REAL) : REAL;
               ASK METHOD IntegrateYb(IN fofYb : REAL; IN N : INTEGER) : REAL;
               ASK METHOD ForElevationAngle() : REAL;
               ASK METHOD GetNoise(IN noise : REAL) : REAL;
               ASK METHOD FindTheSNRatio(IN TP : LocationRecType; IN PP : LocationRecType;
                                         IN SensParam : SensorInfoRecType;
                                         IN TgTObj : PlatformObj);

          END OBJECT;
VAR
  Environment : EnvironmentObj;  { Need to be newed somewhere}
```

```
    Radar      : RadarObj;       { Need to be newed somewhere}
    Threat     : ThreatObj;      { Need to be newed somewhere}
    Threshold  : REAL;
    k          : REAL;
  END MODULE.
```

Implementation module IAARON1.mod:

```
IMPLEMENTATION MODULE AARON1;

FROM GM          IMPORT LocationRecType;
FROM Plat        IMPORT PlatformObj;
FROM MathMod     IMPORT LOG10, POWER, EXP, LN, pi, COS, SQRT;
{ use this to get info from the platform records}
FROM PList       IMPORT MasterPlatformInfoList, PlatformInfoListObj,
                        PlatformInfoRecType;
FROM DetRng      IMPORT AttributesRecType, MasterSensorList, SensorMasterListObj,
                        SensorInfoRecType, DetectionRangePairObj;
FROM MathFunc    IMPORT MyMath;
FROM Write       IMPORT WriteLine, WriteData;
FROM AARONCDF    IMPORT BlackBox;
FROM AARONTIME   IMPORT MasterEntryTime;
FROM TypeLis     IMPORT MasterPlatformTypeList, PlatformTypeRecType;
FROM Space       IMPORT Sensor;
FROM RandMod     IMPORT RandomObj;
FROM Manuv       IMPORT NamedManeuverObj, ManeuverListObj, NamedPathObj,
                        ManeuveringPlatformObj;
FROM SimpleStats IMPORT StatObj;
FROM SimExp      IMPORT ExperimentManager;

OBJECT ThreatObj;
ASK METHOD ObjInit;
  BEGIN
  END METHOD;

ASK METHOD GiveMeanRCS(IN ThisThreat : STRING): REAL;
VAR
   Rec : PlatformInfoRecType;
   BEGIN
      Rec := ASK MasterPlatformInfoList TO GivePlatform(ThisThreat);
      RETURN(Rec.RCS);
   END METHOD;
END OBJECT;

OBJECT EnvironmentObj;
ASK METHOD ObjInit;
   BEGIN
      ClutterVar := 0.0;
      SystemLoss := 1.26;{Says system losses, 1dB}
      AtmosLoss := 3.16;{Says atmos losses, 5dB}
      NEW(MyRand); {Used for the RCS}
      RandTime := 0.0; {Initialized to zero}
   END METHOD;
```

```
 ASK METHOD GiveRCS(IN NewThreat : STRING): REAL;
    BEGIN
       RETURN(ASK Threat GiveMeanRCS(NewThreat));
    END METHOD;

  ASK METHOD Clutter() : REAL;
     BEGIN
        RETURN(ClutterVar);
     END METHOD;

 ASK METHOD SysLoss() : REAL;
    BEGIN
       RETURN(SystemLoss);
    END METHOD;

  ASK METHOD AtmLoss() : REAL;
     BEGIN
        RETURN(AtmosLoss);
     END METHOD;

 ASK METHOD SetCurrentPlats(IN P : PlatformObj; IN T : PlatformObj);
{This method sets the active participants platform objects}
    BEGIN
       ActiveThreat := T;
       ActivePlat := P;
    END METHOD;

ASK METHOD SetEntryLocation(IN Locale : LocationRecType);
    BEGIN
       FutureLocale := Locale;
    END METHOD;

ASK METHOD FindDetectTime(IN PulseInt : INTEGER;IN ThisPlatID : STRING;
                                    IN ThisSensor : STRING;IN ThisTarget : PlatformObj;
                                    IN Ro : REAL);
VAR
    Time, SlantRangeTime,
    speed                  : REAL;
    temprec                : PlatformTypeRecType;
    IDRec                  : PlatformInfoRecType;
    TrueRange,
    RadianAngle            : REAL;
    Xnot, Znot, a,b,c,VSI  : REAL;
    MyPosit,
    ThreatPosit,ThreatVel  : LocationRecType;
    VelocityZ,VX           : REAL;
    ThisPath               : NamedPathObj;
    TheseManeuvers         : ManeuverListObj;
    NumberOfMan            : INTEGER;
    SensorParameters       : SensorInfoRecType;
    BEGIN
       SensorParameters := ASK MasterSensorList GiveSensor(ThisSensor);
       ASK BlackBox TO SetLookTimes(SensorParameters,{TheseManeuvers,} Ro);
       SlantRangeTime := ASK BlackBox TO FindARange(PulseInt);{Returns random range of detection}
```

```
            IDRec := ASK MasterPlatformInfoList GivePlatform(ThisPlatID);
            {Pass Time to IDetect.mod}
            RandTime := SlantRangeTime;
            {Store times here}
            ASK MasterEntryTime TO AddRecord(IDRec.PlatformName,ThisSensor,
                                                          ASK ThisTarget ID,SlantRangeTime);

        END METHOD;

    ASK METHOD PassDetectionRCS(IN ThisThreat : PlatformObj) : REAL;
    VAR
        muRCS, R    : REAL;
        RCSatDetection : REAL;
        BEGIN
          muRCS := ASK SELF GiveRCS(ASK ThisThreat ID);
          R :=  ASK MyRand UniformReal(0.0,1.0);
          RCSatDetection := -muRCS * LN(R);
          RETURN(RCSatDetection);
        END METHOD;
    END OBJECT;

    OBJECT RadarObj;
    ASK METHOD ObjInit;
       BEGIN
          k := 1.38/POWER(10.0,23.0);
          NEW(RCSStatKeeper);
         NEW(RandVar1);
         ASK RandVar1 TO SetSeed(2116429);
       END METHOD;

    ASK METHOD ThermalVar(IN Csensor : SensorInfoRecType) : REAL;
    VAR
       value : REAL;
       BEGIN
         value := k*Csensor.Attributes.ReceiverTemp*Csensor.Attributes.FilterBW ;
         RETURN(value);
       END METHOD;

    ASK METHOD Interference(IN CSensor : SensorInfoRecType) : REAL;
    VAR
       IntVar : REAL;
       BEGIN
         IntVar := ASK SELF ThermalVar(CSensor) + ASK Environment TO Clutter();
         RETURN(IntVar);
       END METHOD;

    ASK METHOD FindRo(IN TgtType : STRING;
                             IN SensorTypeName : STRING): REAL;
    VAR
       sigma, term1, term2, term3,A : REAL;
       i                  : INTEGER;
       CurrentSensor      : SensorInfoRecType;
       Noise, Clutter     : REAL;
       B                  : INTEGER;
```

```
      BEGIN
         CurrentSensor := ASK MasterSensorList TO GiveSensor(SensorTypeName);
         sigma := ASK Environment TO GiveRCS(TgtType);
         term1 := CurrentSensor.Attributes.AveragePower * CurrentSensor.Attributes.Aperture *
                    ASK SELF Gain(CurrentSensor)*sigma * CurrentSensor.Attributes.Efficiency * 0.5 *
                    FLOAT(CurrentSensor.Attributes.PulseInt);
         term2 := (16.0*POWER(pi,FLOAT(2))) * Interference(CurrentSensor) ;
         term3 := ASK Environment SysLoss() * ASK Environment AtmLoss();
         B := CurrentSensor.Attributes.PulseInt;
         A := CurrentSensor.Attributes.FalseAlarmNumber;
         Yb := ASK SELF FindBiasLevel(B, A); {Sets the Threshold}
         {Returns the Max det range based on RCS and radar parameters}
         RETURN(POWER(term1/(term2 * term3),0.25) / 1852.0); {Convert from meters}
      END METHOD;

   ASK METHOD Gain(IN ThisSensor : SensorInfoRecType): REAL;
      {Need to pass the actual record}
   VAR
      ap     : REAL;
      lambda : REAL;
      BEGIN
         ap := ThisSensor.Attributes.Aperture;
         lambda := ThisSensor.Attributes.WaveLength;
         RETURN((4.0 * pi * ap)/POWER(lambda,FLOAT(2)));
      END METHOD;

  ASK METHOD FindBiasLevel(IN b : INTEGER; IN NPrime : REAL) : REAL;
     { NPrime= FALSE ALARM NUMBER, b= PULSES TO INTEGRATE}
  VAR
     numerator, denom1, denom2 : REAL;
     a, C : REAL;
     BEGIN
        a := POWER(0.5, (1.0/NPrime)); {This is the value to check against}
         {a is FofYb}
        C := ASK SELF TO IntegrateYb(a, b);{Pass FofYb and Pulses to Integrate}
        RETURN(C);
     END METHOD;

 ASK METHOD IntegrateYb(IN fofYb : REAL; IN N : INTEGER) : REAL;
    {Integrate For the purpose of finding the Threshold}
 VAR
    numerator, denom1, denom2      : REAL;
    a, b, h, temp, sum, X, dumy    : REAL;
    answernew, y, delta            : REAL;
    FoundSolution                  : BOOLEAN;
    littlen, i                     :INTEGER;
    BEGIN
       littlen := 800;
       FoundSolution := FALSE;
       a:= 0.0;
       b:= 0.1;
       h:= (b-a)/FLOAT(littlen);
       temp := (EXP(-1.0*b)) * (POWER(b, FLOAT(N - 1))); {sum of f(a) and f(b)}
       sum := ASK MyMath TO ComputeFactorial(N - 1, temp);{this is the last term}
```

```
        FOR i:= 1 TO (littlen - 1)
          X:= h * FLOAT(i);
          dumy:= (EXP(-1.0 * X)) * (POWER(X, FLOAT(N-1))) ;{2 times y}
           y := ASK MyMath TO ComputeFactorial(N - 1,dumy);
           sum:= sum + (2.0 * y);
        END FOR;
        answernew := (h/2.0) * sum;
        delta := 0.1;  { This is the increase in Yb until the right value is found }
        REPEAT
          b := b + delta;
          h:= (b-a)/FLOAT(littlen);
          temp := (EXP(-1.0*b)) * (POWER(b,FLOAT(N-1)));  {sum of f(a) and f(b)}
          sum := ASK MyMath TO ComputeFactorial(N - 1, temp);
          FOR i:= 1 TO (littlen - 1)
            X:= h * FLOAT(i);
            dumy:= (EXP(-1.0*X)) * (POWER(X,FLOAT(N-1))) ;{2 times y}
             y :=  ASK MyMath TO ComputeFactorial(N - 1,dumy);
            sum:= sum + (2.0 * y);
          END FOR;
         answernew := (h/2.0) * sum;
         IF fofYb < answernew
            Threshold := b;
            FoundSolution := TRUE;
            RETURN(b); {This b is the thresholded value}
         END IF;
      UNTIL FoundSolution;
   END METHOD;

ASK METHOD ForElevationAngle() : REAL;
VAR
    Angle  : REAL;
   BEGIN
      NEW(Sensor);
      Angle := ASK Sensor TO GiveElevationAngle();
      DISPOSE(Sensor);
      RETURN(Angle);
   END METHOD;

ASK METHOD GetNoise(IN noise : REAL) : REAL;
  {Uses the invers transform of the Weibull}
VAR
    U , Beta,
    RandWeibull : REAL;

    BEGIN
       U := ASK RandVar1 UniformReal(0.0,1.0);
       Beta := SQRT(2.0*noise);
       RandWeibull := Beta*POWER(-LN(U),0.5);
       RETURN(RandWeibull);
   END METHOD;

ASK METHOD FindTheSNRatio(IN TP : LocationRecType; IN PP : LocationRecType;
                              IN SensParam : SensorInfoRecType;
                              IN TgTObj : PlatformObj);
    {For analysis purposes find the S/N Ratio}
  VAR
     term1, term2,
     term3, term4,
     SNRatio      : REAL;
     TheRange     : REAL;
     DetRCS       : REAL;
     TheNoise,
     TheSignal,SplusN      : REAL;
     BEGIN
        TheRange := SQRT(POWER(TP.x-PP.x,2.0) + POWER(TP.y-PP.y,2.0) +
                           POWER(TP.z-PP.z,2.0)) * 1852.0; {Convert units to meter}
        DetRCS := ASK Environment TO PassDetectionRCS(TgTObj);{Here use rand RCS}
        term1 := SensParam.Attributes.AveragePower*Gain(SensParam)*DetRCS*
                 SensParam.Attributes.Aperture*SensParam.Attributes.Efficiency*
                 FLOAT(SensParam.Attributes.PulseInt);
        term2 := POWER((4.0*pi*POWER(TheRange,2.0)),2.0);
        term3 := Interference(SensParam);
        term4 := ASK Environment SysLoss() * ASK Environment AtmLoss();
        SNRatio := term1/(term2 * term3 * term4);
        TheNoise := GetNoise(term3);
        TheSignal := term1/(term2 * term4);
        SplusN := (TheSignal + TheNoise)*POWER(10.0,14.0);
        WriteData(TheSignal*POWER(10.0,14.0));
        ASK RCSStatKeeper TO GetSample(TheSignal*POWER(10.0,14.0)); {Grab Stats}
        IF ((ASK RCSStatKeeper HW() / ASK RCSStatKeeper Mean()) <=
                      ASK ExperimentManager DesiredPrecision) AND (ASK RCSStatKeeper N > 1)
           ASK RCSStatKeeper TO Output;
           ASK ExperimentManager TO SetBoolean;
        END IF;
   END METHOD;
END OBJECT;
END MODULE.
```

IAARONCDF.mod builds the cumulative distribution function from which the inverse transform method determines the random time of detection.

Definition module DAARONCDF.mod:

```
DEFINITION MODULE AARONCDF;

FROM RandMod IMPORT RandomObj;
FROM Manuv   IMPORT ManeuverListObj;
FROM DetRng  IMPORT AttributesRecType,SensorInfoRecType;
FROM GM      IMPORT LocationRecType;

{ THE PURPOSE OF THIS MODULE IS TO GENERATE THE RV Time}
TYPE
   CDFRec = RECORD
                 Range    : REAL;
                 NoDetect : REAL;
                 CumPd    : REAL;
```

```
                    PDF      : REAL;
                 END RECORD;
      CDFObj = OBJECT
                    Pd          : ARRAY INTEGER OF REAL;
                    Ufour       : ARRAY INTEGER OF REAL;
                    CumProb     : ARRAY INTEGER OF CDFRec;
                    Ro          : REAL;
                    RandVar     : RandomObj;
                    Ratio       : ARRAY INTEGER OF REAL;
                    LookRanges  : ARRAY INTEGER OF REAL;
                    TimeToRange : ARRAY INTEGER OF REAL;
                    K           : INTEGER;

                    ASK METHOD ObjInit;
                    ASK METHOD FindPosTime(IN CSpeed : LocationRecType;
                                           IN TPos   : LocationRecType;
                                           IN PPos   : LocationRecType;
                                           IN Rng    : REAL) : REAL;
                    ASK METHOD SetLookTimes(IN sensorrec : SensorInfoRecType;
                                            IN NewRo : REAL );
                    ASK METHOD FindARange(IN N : INTEGER) : REAL;
                    ASK METHOD FindSingleHitPd(IN N : INTEGER) : REAL;
                    ASK METHOD BuildCDF(IN N : INTEGER) : REAL;
                    ASK METHOD FindRandRange : REAL;
                END OBJECT;
   CONST Partitions = 900; {Set intervals for max number of looks}
   VAR
      BlackBox : CDFObj;
   END MODULE.
```

Implementation module IAARONCDF.mod:

```
IMPLEMENTATION MODULE AARONCDF;

FROM MathMod   IMPORT POWER, FLOOR, EXP, SQRT, SIN, COS, pi, e;
FROM AARON1    IMPORT Environment, Threshold;
FROM RandMod   IMPORT RandomObj;
FROM Manuv     IMPORT ManeuverListObj, NamedManeuverObj;
FROM DetRng    IMPORT AttributesRecType,SensorInfoRecType;
FROM GM        IMPORT LocationRecType;
FROM AngleU    IMPORT RadToDeg, RadEastToDegNorth, BearingToLocation;
FROM MathFunc  IMPORT MyMath;
FROM PList     IMPORT MasterPlatformInfoList, PlatformInfoListObj,
                      PlatformInfoRecType;

OBJECT CDFObj;
ASK METHOD ObjInit;
VAR
   i : INTEGER;
   BEGIN
      NEW(RandVar);
      ASK RandVar TO SetSeed(2116429302);
   END METHOD;
```

```
ASK METHOD FindPosTime( IN RSpeed : LocationRecType;{Relative Speed}
                        IN TPos   : LocationRecType;
                        IN PPos   : LocationRecType;
                        IN Rng    : REAL) : REAL;
VAR
   a, b, c, time : REAL;
   BEGIN
      a := POWER(RSpeed.x,2.0) + POWER(RSpeed.y,2.0) + POWER(RSpeed.z,2.0);
      b := 2.0*((TPos.x*RSpeed.x) + (TPos.y*RSpeed.y) + (TPos.z*RSpeed.z) +
                  (-PPos.x*RSpeed.x) + (-PPos.y*RSpeed.y)  + (-PPos.z*RSpeed.z));
      c := POWER(TPos.x,2.0) + POWER(TPos.y,2.0) + POWER(TPos.z,2.0) +
           POWER(PPos.x,2.0) + POWER(PPos.y,2.0) + POWER(PPos.z,2.0) -
           2.0*((TPos.x*PPos.x) + (TPos.y*PPos.y) + (TPos.z*PPos.z)) - POWER(Rng,2.0);
      time := ASK MyMath Quad(a,b,c);
      RETURN(time);
   END METHOD;

ASK METHOD SetLookTimes(IN sensorrec : SensorInfoRecType;
                        IN NewRo : REAL );
VAR
   i, j                    : INTEGER;
   curRo, nextRange,
   ElaspedTime, TgtCourse,
   PCourse, RangeCheck,
   x, y, z, t, v           : REAL;
   velocity, curBigXY,
   curBigZ, DestBigXY,
   DestBigZ, Bearing,
   NextBearing, speedZ,
   speedXY, TimeInt        : REAL;
   curmanuv,
   nextmanuv, pnmanuv      : NamedManeuverObj;
   PlatCenter,
   curTgtPosit,
   DestTgtPosit, TDestTgtPosit,
   IPTgt, IPPlat,
   MaxTgtPosit,
   PlatSpeed, future,
   SpeedVec, TgtSpeed,
   RoPosit , PDest         : LocationRecType;
   ProperInterval,
   Inbound                 : BOOLEAN;
   counter                 : INTEGER;
   TheseAttributes         : AttributesRecType;
   TgtManuv, Pmanuv        : ManeuverListObj;
   Ttemprec,Ptemprec       :  PlatformInfoRecType;
   BEGIN
      TgtManuv := ASK Environment.ActiveThreat.CurrentPath ManeuverList;
      Ttemprec := ASK MasterPlatformInfoList GivePlatform(ASK Environment.ActiveThreat ID);
      NEW(IPTgt);
      IPTgt.x :=  Ttemprec.InitialLocation.x;
      IPTgt.y :=  Ttemprec.InitialLocation.y;
      IPTgt.z :=  Ttemprec.InitialLocation.z;
      counter := 0;
```

```
 Ro := NewRo;
 TimeInt := sensorrec.Attributes.ScanTime;{Time allowed for movement between scans}
 curmanuv := ASK TgtManuv First();{Always at least 1 maneuver}
 curTgtPosit := IPTgt;
 nextmanuv := ASK TgtManuv Next(curmanuv);
  { *** Plat is not moving so find its first speed}
 NEW(PlatSpeed);
 Ptemprec := ASK MasterPlatformInfoList GivePlatform(ASK Environment.ActivePlat ID);
 PlatCenter :=  Ptemprec.InitialLocation;
 Pmanuv := ASK Environment.ActivePlat.CurrentPath ManeuverList;
 pnmanuv := ASK Pmanuv First();
 PDest := ASK pnmanuv DestinationForStraightShot;
 PCourse := BearingToLocation(PlatCenter, PDest);
 PlatSpeed.z := (ASK pnmanuv ClimbDiveRate)/6000.0;
 velocity := ASK pnmanuv NewSpeed;
 speedXY := SQRT(POWER(velocity,2.0) - POWER(speedZ,2.0));
 PlatSpeed.x := speedXY * SIN(PCourse * pi/180.0);
 PlatSpeed.y := speedXY * COS(PCourse * pi/180.0);
 Inbound := TRUE;
 Bearing := BearingToLocation(PlatCenter, IPTgt);{Assumes Straight Shot}
 ProperInterval := FALSE;
 NEW(future);
 NEW(SpeedVec);
 NEW(TgtSpeed);
 NEW(RoPosit);
 NEW(DestTgtPosit);
 NEW(LookRanges,1..Partitions);
 NEW(TimeToRange,1..Partitions);

  { *** Find the correct starting interval}
 WHILE (curmanuv <> NILOBJ) AND (Inbound)
  {Sets the climb/descent speed and converts to Mile per min, ML in man.dat doesn't work}
 curBigXY := SQRT(POWER((curTgtPosit.x - PlatCenter.x),2.0) +
                  POWER((curTgtPosit.y - PlatCenter.y),2.0));
 curBigZ := curTgtPosit.z - PlatCenter.z ;

  { **** This is for true destination }
 TDestTgtPosit := ASK curmanuv DestinationForStraightShot;

  { ***Sets the inbound speed Vector of the target}
 TgtCourse := BearingToLocation(curTgtPosit, TDestTgtPosit);
 TgtSpeed.z := (ASK curmanuv ClimbDiveRate)/6000.0;
 velocity := ASK curmanuv NewSpeed;
 speedXY := SQRT(POWER(velocity,2.0) - POWER(speedZ,2.0));
 TgtSpeed.x := speedXY * SIN(TgtCourse * pi/180.0);
 TgtSpeed.y := speedXY * COS(TgtCourse * pi/180.0);

 { *** Sets the Rate of Closure Vector }
 SpeedVec.x := TgtSpeed.x - PlatSpeed.x;
 SpeedVec.y := TgtSpeed.y - PlatSpeed.y;
 SpeedVec.z := TgtSpeed.z - PlatSpeed.z;

 { Rework and solve for relative destination and check, v is the time to get to
                                   true target destination from current TgtPosit}
```

```
v := SQRT(POWER(TDestTgtPosit.x - curTgtPosit.x,2.0) +
            POWER(TDestTgtPosit.y - curTgtPosit.y,2.0) +
            POWER(TDestTgtPosit.y - curTgtPosit.y,2.0)) / velocity;
IF nextmanuv <> NILOBJ
DestTgtPosit.x := curTgtPosit.x + (SpeedVec.x*v);
DestTgtPosit.y := curTgtPosit.y + (SpeedVec.y*v);
DestTgtPosit.z := curTgtPosit.z + (SpeedVec.z*v);
DestBigXY := SQRT(POWER((DestTgtPosit.x - PlatCenter.x),2.0) +
                       POWER((DestTgtPosit.y - PlatCenter.y),2.0));
DestBigZ := DestTgtPosit.z - PlatCenter.z;
ELSE
   DestTgtPosit.x := TDestTgtPosit.x;
   DestTgtPosit.y := TDestTgtPosit.y ;
   DestTgtPosit.z := TDestTgtPosit.z;
   DestBigXY := SQRT(POWER((DestTgtPosit.x - PlatCenter.x),2.0) +
                          POWER((DestTgtPosit.y - PlatCenter.y),2.0));
   DestBigZ := DestTgtPosit.z - PlatCenter.z;
END IF;
IF (SQRT(POWER(curBigXY,2.0) + POWER(curBigZ,2.0)) <=
                       SQRT(POWER(DestBigXY,2.0) + POWER(DestBigZ,2.0))) AND (Inbound)
   MaxTgtPosit := curTgtPosit;
   { *** t is the time to get to Ro from MaxTgtPosit}
   t := FindPosTime(SpeedVec, MaxTgtPosit, PlatCenter, Ro);
   { *** Find the coordinates of Ro }
   RoPosit.x := MaxTgtPosit.x + (SpeedVec.x*t);
   RoPosit.y := MaxTgtPosit.y + (SpeedVec.y*t);
   RoPosit.z := MaxTgtPosit.z + (SpeedVec.z*t);
   RangeCheck := SQRT(POWER(curBigXY,2.0) + POWER(curBigZ,2.0));
   ProperInterval := TRUE;
   REPEAT
      future.x := curTgtPosit.x + (SpeedVec.x*TimeInt);{New x posit}
      future.y := curTgtPosit.y + (SpeedVec.y*TimeInt);{New y posit}
      future.z := curTgtPosit.z + (SpeedVec.z*TimeInt);{New z posit}
      nextRange := SQRT(POWER(future.x - PlatCenter.x,2.0) +
                             POWER(future.y - PlatCenter.y,2.0) +
                             POWER(future.z - PlatCenter.z,2.0));
      curTgtPosit := future;
       { Check if the range is decreasing, if so, add to array if R < Ro}
      IF nextRange <= RangeCheck
         curBigXY := SQRT(POWER((curTgtPosit.x - PlatCenter.x),2.0) +
                              POWER((curTgtPosit.y - PlatCenter.y),2.0));
         curBigZ := curTgtPosit.z - PlatCenter.z;
         RangeCheck := nextRange;
         IF nextRange <= Ro
           counter := counter + 1;
           LookRanges[counter] := nextRange;
            { *** Time to go from Ro to all other accepted positions }
           IF counter <> 1
             TimeToRange[counter] := (SQRT(POWER(future.x - RoPosit.x,2.0)
                                          + POWER(future.y - RoPosit.y,2.0) +
                                            POWER(future.z - RoPosit.z,2.0))
                                 /   (SQRT(POWER(SpeedVec.x ,2.0) +
                                           POWER(SpeedVec.y,2.0) +
                                           POWER(SpeedVec.z,2.0)))) +
                    TimeToRange[counter - 1] ;
                    RoPosit.x := future.x;
                    RoPosit.y := future.y;
                    RoPosit.z := future.z;
                  ELSE
                  { *** Time to go from Ro to first accepted position}
                  TimeToRange[counter] := SQRT(POWER(future.x - RoPosit.x,2.0)
                                               + POWER(future.y - RoPosit.y,2.0)
                                               + POWER(future.z - RoPosit.z,2.0))
                                           / SQRT(POWER(SpeedVec.x ,2.0) +
                                                  POWER(SpeedVec.y,2.0) +
                                                  POWER(SpeedVec.z,2.0));
                  RoPosit.x := future.x;
                  RoPosit.y := future.y;
                  RoPosit.z := future.z;
                END IF;
              END IF;
            ELSE
              Inbound := FALSE;{Tgt passed ovhd}
              curmanuv := nextmanuv;
            END IF;
          UNTIL NOT(Inbound);
            {Force the target outbound after min range}
            Inbound := FALSE;{Tgt passed ovhd}
            { Removed while loop because since in this manuever the target will pass
                                               ovhd you do not want to consider a next manuever}
            {for when Max is less than Min, the normal configuration execute the following}
        ELSIF (Ro > SQRT(POWER(DestBigXY,2.0) + POWER(DestBigZ,2.0))) AND (Inbound)
          MaxTgtPosit := curTgtPosit;

          { *** t is the time to get to Ro from MaxTgtPosit}
          t := FindPosTime(SpeedVec, MaxTgtPosit, PlatCenter, Ro);

          { *** Find the coordinates of Ro }
          RoPosit.x := MaxTgtPosit.x + (SpeedVec.x*t);
          RoPosit.y := MaxTgtPosit.y + (SpeedVec.y*t);
          RoPosit.z := MaxTgtPosit.z + (SpeedVec.z*t);
          RangeCheck := SQRT(POWER(curBigXY,2.0) + POWER(curBigZ,2.0));
          ProperInterval := TRUE;
          REPEAT
            future.x := curTgtPosit.x + (SpeedVec.x*TimeInt);{New x posit}
            future.y := curTgtPosit.y + (SpeedVec.y*TimeInt);{New y posit}
            future.z := curTgtPosit.z + (SpeedVec.z*TimeInt);{New z posit}
            nextRange := SQRT(POWER(future.x - PlatCenter.x,2.0) +
                              POWER(future.y - PlatCenter.y,2.0) +
                              POWER(future.z - PlatCenter.z,2.0));
            curTgtPosit := future;

            { *** Check if the range is decreasing, if so, add to array if R < Ro}
            IF nextRange <= RangeCheck
              curBigXY := SQRT(POWER((curTgtPosit.x - PlatCenter.x),2.0) +
                               POWER((curTgtPosit.y - PlatCenter.y),2.0));
              curBigZ := curTgtPosit.z - PlatCenter.z;
              RangeCheck := nextRange;
            IF nextRange <= Ro
               counter := counter + 1;
               LookRanges[counter] := nextRange;

              { *** Time to go from Ro to all other accepted positions }
              IF counter <> 1
                 TimeToRange[counter] := (SQRT(POWER(future.x - RoPosit.x,2.0)
                                                + POWER(future.y - RoPosit.y,2.0)
                                                + POWER(future.z - RoPosit.z,2.0))
                                        / SQRT(POWER(SpeedVec.x ,2.0) +
                                                 POWER(SpeedVec.y,2.0) +
                                                 POWER(SpeedVec.z,2.0)))
                                                              + TimeToRange[counter - 1] ;
                 RoPosit.x := future.x;
                 RoPosit.y := future.y;
                 RoPosit.z := future.z;
             ELSE
                 { *** Time to go from Ro to first accepted position}
                 TimeToRange[counter] := SQRT(POWER(future.x - RoPosit.x,2.0)
                                               + POWER(future.y - RoPosit.y,2.0)
                                               + POWER(future.z - RoPosit.z,2.0))
                                       / SQRT(POWER(SpeedVec.x ,2.0) +
                                                POWER(SpeedVec.y,2.0) +
                                                POWER(SpeedVec.z,2.0));
                 RoPosit.x := future.x;
                 RoPosit.y := future.y;
                 RoPosit.z := future.z;
            END IF;
         END IF;
      ELSE
         Inbound := FALSE;{Tgt passed ovhd}
         curmanuv := nextmanuv;
      END IF;
   UNTIL (nextRange < SQRT(POWER(DestBigXY,2.0) + POWER(DestBigZ,2.0))) OR (NOT(Inbound));
   WHILE nextmanuv <> NILOBJ  {Move to next Maneuver}
      MaxTgtPosit := curTgtPosit;
      curmanuv := nextmanuv;
      nextmanuv := ASK TgtManuv Next(curmanuv);
      {  **** This is for true destination }
      TDestTgtPosit := ASK curmanuv DestinationForStraightShot;
      TgtCourse := BearingToLocation(curTgtPosit, TDestTgtPosit);
      TgtSpeed.z := (ASK curmanuv ClimbDiveRate)/6000.0;
      velocity := ASK curmanuv NewSpeed;
      speedXY := SQRT(POWER(velocity,2.0) - POWER(speedZ,2.0));
      TgtSpeed.x := speedXY * SIN(TgtCourse * pi/180.0);
      TgtSpeed.y := speedXY * COS(TgtCourse * pi/180.0);
      SpeedVec.x := TgtSpeed.x - PlatSpeed.x;
      SpeedVec.y := TgtSpeed.y - PlatSpeed.y;
      SpeedVec.z := TgtSpeed.z - PlatSpeed.z;
      v := SQRT(POWER(TDestTgtPosit.x - curTgtPosit.x,2.0) +
                POWER(TDestTgtPosit.y - curTgtPosit.y,2.0) +
                POWER(TDestTgtPosit.y - curTgtPosit.y,2.0)) / velocity;
      IF nextmanuv <> NILOBJ
         DestTgtPosit.x := curTgtPosit.x + (SpeedVec.x*v);
```

```
    DestTgtPosit.y := curTgtPosit.y + (SpeedVec.y*v);
    DestTgtPosit.z := curTgtPosit.z + (SpeedVec.z*v);
    DestBigXY := SQRT(POWER((DestTgtPosit.x - PlatCenter.x),2.0) +
                      POWER((DestTgtPosit.y - PlatCenter.y),2.0));
    DestBigZ := DestTgtPosit.z - PlatCenter.z;
ELSE
    DestTgtPosit.x := TDestTgtPosit.x;
    DestTgtPosit.y := TDestTgtPosit.y ;
    DestTgtPosit.z := TDestTgtPosit.z;
    DestBigXY := SQRT(POWER((DestTgtPosit.x - PlatCenter.x),2.0) +
                      POWER((DestTgtPosit.y - PlatCenter.y),2.0));
    DestBigZ := DestTgtPosit.z - PlatCenter.z;
END IF;
RangeCheck := SQRT(POWER(curBigXY,2.0) + POWER(curBigZ,2.0));
REPEAT
    future.x := curTgtPosit.x + (SpeedVec.x*TimeInt);{New x pos}
    future.y := curTgtPosit.y + (SpeedVec.y*TimeInt);{New y pos}
    future.z := curTgtPosit.z + (SpeedVec.z*TimeInt);{New z pos}
    nextRange := SQRT(POWER(future.x - PlatCenter.x,2.0) +
                      POWER(future.y - PlatCenter.y,2.0) +
                      POWER(future.z - PlatCenter.z,2.0));
    curTgtPosit := future;
    { *** Check if the range is decreasing, if so, add to array if R < Ro}
    IF nextRange <= RangeCheck
        curBigXY := SQRT(POWER((curTgtPosit.x - PlatCenter.x),2.0) +
                         POWER((curTgtPosit.y - PlatCenter.y),2.0));
        curBigZ := curTgtPosit.z - PlatCenter.z;
        RangeCheck := nextRange;
        IF nextRange <= Ro
            counter := counter + 1;
            LookRanges[counter] := nextRange;
            IF counter <> 1
            TimeToRange[counter] := (SQRT(POWER(future.x - RoPosit.x,2.0)
                                       + POWER(future.y - RoPosit.y,2.0)
                                       + POWER(future.z - RoPosit.z,2.0))
                                   / SQRT(POWER(SpeedVec.x ,2.0) +
                                       POWER(SpeedVec.y,2.0) +
                                       POWER(SpeedVec.z,2.0)))
                                                   + TimeToRange[counter - 1] ;
            RoPosit.x := future.x;
            RoPosit.y := future.y;
            RoPosit.z := future.z;
        ELSE
            { *** Time to go from Ro to first accepted position}
            TimeToRange[counter] := SQRT(POWER(future.x - RoPosit.x,2.0)
                                      + POWER(future.y - RoPosit.y,2.0)
                                      + POWER(future.z - RoPosit.z,2.0))
                                  / SQRT(POWER(SpeedVec.x ,2.0) +
                                      POWER(SpeedVec.y,2.0) +
                                      POWER(SpeedVec.z,2.0));
            RoPosit.x := future.x;
            RoPosit.y := future.y;
            RoPosit.z := future.z;
        END IF;
```

```
            END IF;
         ELSE
            Inbound := FALSE;{Tgt passed ovhd}
            curmanuv := nextmanuv;
         END IF;
   UNTIL (nextRange < SQRT(POWER(DestBigXY,2.0) + POWER(DestBigZ,2.0))) OR (NOT(Inbound));
 END WHILE;
     {Backup Plan for a fly through situation}
     IF Inbound
        RangeCheck := SQRT(POWER(curBigXY,2.0) + POWER(curBigZ,2.0));
        REPEAT
           future.x := curTgtPosit.x + (SpeedVec.x*TimeInt);{New x posit}
           future.y := curTgtPosit.y + (SpeedVec.y*TimeInt);{New y posit}
           future.z := curTgtPosit.z + (SpeedVec.z*TimeInt);{New z posit}
           nextRange := SQRT(POWER(future.x - PlatCenter.x,2.0) +
                             POWER(future.y - PlatCenter.y,2.0) +
                             POWER(future.z - PlatCenter.z,2.0));
           curTgtPosit := future;
           { Check if the range is decreasing, if so, add to array if R < Ro}
           IF nextRange <= RangeCheck
              curBigXY := SQRT(POWER((curTgtPosit.x - PlatCenter.x),2.0) +
                               POWER((curTgtPosit.y - PlatCenter.y),2.0));
              curBigZ := curTgtPosit.z - PlatCenter.z;
              RangeCheck := nextRange;
              IF nextRange <= Ro
                 counter := counter + 1;
                 LookRanges[counter] := nextRange;
                  { *** Time to go from Ro to all other accepted positions }
                 IF counter <> 1
                    TimeToRange[counter] := (SQRT(POWER(future.x - RoPosit.x,2.0)
                                              + POWER(future.y - RoPosit.y,2.0) +
                                                POWER(future.z - RoPosit.z,2.0))
                                        /   (SQRT(POWER(SpeedVec.x ,2.0) +
                                                POWER(SpeedVec.y,2.0) +
                                                POWER(SpeedVec.z,2.0)))) +
                     TimeToRange[counter - 1] ;
                     RoPosit.x := future.x;
                    RoPosit.y := future.y;
                    RoPosit.z := future.z;
                 ELSE
                    { *** Time to go from Ro to first accepted position}
                    TimeToRange[counter] := SQRT(POWER(future.x - RoPosit.x,2.0)
                                            + POWER(future.y - RoPosit.y,2.0)
                                            + POWER(future.z - RoPosit.z,2.0))
                                         / SQRT(POWER(SpeedVec.x ,2.0) +
                                              POWER(SpeedVec.y,2.0) +
                                              POWER(SpeedVec.z,2.0));
                    RoPosit.x := future.x;
                    RoPosit.y := future.y;
                    RoPosit.z := future.z;
                 END IF;
              END IF;
           ELSE
              Inbound := FALSE;{Tgt passed ovhd}
```

```
                curmanuv := nextmanuv;
             END IF;
          UNTIL NOT(Inbound);
      END IF;
   ELSE { *** Check the next set of points (main if loop)}
      curmanuv := nextmanuv;
      curTgtPosit.x := DestTgtPosit.x;
      curTgtPosit.y := DestTgtPosit.y;
      curTgtPosit.z := DestTgtPosit.z;
      IF curmanuv <> NILOBJ
         nextmanuv := ASK TgtManuv Next(curmanuv);
      END IF;
   END IF; {main if loop}
   IF NOT(Inbound)
      { *** The array is stored in reverse order here}
      K := counter; {Lets me know the size of the array}
      EXIT;
   END IF;
   END WHILE;
   DISPOSE(future);
   DISPOSE(SpeedVec);
   DISPOSE(TgtSpeed);
   DISPOSE(RoPosit);
   DISPOSE(DestTgtPosit);
   DISPOSE(PlatSpeed);
   DISPOSE(IPTgt);
   END METHOD;

   ASK METHOD FindARange(IN N : INTEGER) : REAL;
   VAR
      a, b, answer  : REAL;
      i             : INTEGER;
      BEGIN
         NEW(Ufour, 1..K);
         a := (Threshold - FLOAT(N) + 1.0)/FLOAT(N);
          { *** LookRanges goes from [1] = furthest out range from platform[k] = closest range to platform.
                The ratio of R/Ro, K intervals. This is an array of U^4th power}
         FOR i := 1 TO K
            Ufour[i] := a * POWER((LookRanges[i] / Ro), 4.0);
         END FOR;
         answer := ASK SELF TO FindSingleHitPd(N);
         DISPOSE(Ufour);
         DISPOSE(Pd);
         DISPOSE(CumProb);
         DISPOSE(Ratio);
         DISPOSE(LookRanges);
         DISPOSE(TimeToRange);
         RETURN(answer);
      END METHOD;

   ASK METHOD FindSingleHitPd(IN N : INTEGER) : REAL;
   VAR
      i         : INTEGER;
      answer    : REAL;
```

```
    BEGIN
      NEW(Pd, 1..K);
       {Array of The Single Hit Probability of Detection}
      FOR i := 1 TO K
         Pd[i] := EXP(-Ufour[i]);
      END FOR;
      answer := ASK SELF TO BuildCDF(N);
      RETURN(answer);
    END METHOD;

ASK METHOD BuildCDF(IN N : INTEGER) : REAL;
VAR
    Check, CumValue,
    a, b, CumSum, c,
    h1,sum              : REAL;
    answer              : REAL;
    cdfrec              : CDFRec;
    i, j                : INTEGER;
    BEGIN
      NEW(Ratio, 1..K);
       { *** Find the R/Ro ratio and store it}
      FOR i := 1 TO K
         Ratio[i] := LookRanges[i] / Ro;
      END FOR;
      { ***  Now Find 1 - SHPD and load to matrix  }
      NEW(CumProb, 1..K);
      FOR i := 1 TO K
         NEW(CumProb[i]);
         CumProb[i].Range := LookRanges[i];
         { *** Find the Appropriate Single Hit Pd and find prob no detect at look i}
        CumProb[i].NoDetect := 1.0 - Pd[i]; {Probability of no detection}
     END FOR;
     { *** Finally compute the cumulative Pd}
     { *** Initialize}
     CumProb[1].CumPd := Pd[1];
     CumSum := CumProb[1].CumPd;
     OUTPUT("CDF       Range        Time        Index");
     OUTPUT(CumSum,"    ",CumProb[1].Range,"     ",TimeToRange[1],"     1" );  }
     FOR i := 2 TO K
        CumValue := 1.0;
        FOR j := 1 TO i
           IF j = i
              CumValue := CumValue * Pd[j];
           ELSE
              CumValue := CumValue * CumProb[j].NoDetect;
           END IF;
        END FOR;
        CumSum := CumSum + CumValue;
        CumProb[i].CumPd := CumSum;  {Cummulative Cdf IS BUILT}
        OUTPUT(CumProb[i].CumPd,"    ",CumProb[i].Range,"     ",TimeToRange[i],             "     ",i);
     END FOR;
     { *** In case CDF does not go to 1, then rescale such that it will be a CDF}
     IF (CumProb[K].CumPd < 1.0 + e) AND (CumProb[K].CumPd > 1.0 - e)
        FOR i := 1 TO K
```

```
            CumProb[i].CumPd := CumProb[i].CumPd / CumProb[K].CumPd;
          END FOR;
        END IF;
        answer := ASK SELF TO FindRandRange;
        RETURN(answer);
      END METHOD;

  ASK METHOD FindRandRange : REAL;
  {Inverse transform method pg. 470 for discrete emperical distribution}
  VAR
      SetU, P          : REAL;
      I, i, j          : INTEGER;
      SelectedRange    : REAL;
      FoundI           : BOOLEAN;
      BEGIN
        FoundI := FALSE;
        i := 1;
        SetU := ASK RandVar UniformReal(0.0,1.0);
        REPEAT
          IF (i = 1) AND (SetU <= CumProb[i].CumPd)
            SelectedRange := TimeToRange[i];
            FoundI := TRUE;
          ELSIF (SetU > CumProb[i].CumPd) AND (SetU <= CumProb[i+1].CumPd)
            SelectedRange := TimeToRange[i+1];
            FoundI := TRUE;
          ELSE
            i := i+1;
          END IF;
        UNTIL FoundI = TRUE;
        RETURN(SelectedRange);
      END METHOD;
END OBJECT;
END MODULE.
```

The module *IAARONTIME.mod* stores the random time of detection, the platform and target combination, and the radar sensor system that is detecting the target.

Definition module DAARONTIME.mod:

```
DEFINITION MODULE AARONTIME;

FROM RandMod IMPORT RandomObj;
FROM  ListMod IMPORT QueueList;
{ THE PURPOSE OF THIS MODULE IS TO HOLD ALL ENTRY DETECTION TIMES}
TYPE
   TimeRec = RECORD
                    Plat      : STRING;
                    Sensor    : STRING;
                    Threat    : STRING;
                    Time      : REAL;
                 END RECORD;
```

```
    TimeObj = OBJECT(QueueList)
                    temp : TimeRec;

                    ASK METHOD ObjInit;
                    ASK METHOD AddRecord(IN P : STRING;IN S : STRING;
                                            IN T : STRING;IN Time : REAL);
                    ASK METHOD EmptyMe;
                 END OBJECT;
VAR
    MasterEntryTime : TimeObj;
END MODULE.
```

Implementation module IAARONTIME.mod:

```
IMPLEMENTATION MODULE AARONTIME;
FROM RandMod IMPORT RandomObj;

OBJECT TimeObj;
ASK METHOD ObjInit;
   BEGIN
   END METHOD;

ASK METHOD AddRecord(IN P : STRING;IN S : STRING;IN T : STRING;IN Time : REAL);
   BEGIN
      NEW(temp);
      temp.Plat := P;
      temp.Sensor := S;
      temp.Threat := T;
      temp.Time := Time;
      ASK SELF TO Add(temp);
   END METHOD;

ASK METHOD EmptyMe;
VAR
   ThisRec, NextRec : TimeRec;
   I, J             : INTEGER;
   BEGIN
      J := ASK SELF numberIn;
      ThisRec := ASK SELF First;
      NextRec := ASK SELF Next(ThisRec);
      FOR I := 1 TO J
         IF I <> J
            ASK SELF TO RemoveThis(ThisRec);
            ThisRec := NextRec;
            NextRec := ASK SELF Next(ThisRec);
         ELSE
            ASK SELF TO RemoveThis(ThisRec);
         END IF;
      END FOR;
   END METHOD;
END OBJECT;
END MODULE.
```

The module *IDetect.mod* is the engine for all detections that take place in a given scenario. It uses the cookie cutter model to determine the time of detection. Changes were made to the module so that if a sensor of type 'RADAR' is being evaluated then apply the random method of determining a detection time. If not, use the standard cookie cutter model. The listed code is an example of how the switching between random and deterministic routines is accomplished. Similar changes were also made in the method SolveCurvedProblem of *IDetect.mod.*

Example from IDetect.mod ASK METHOD StraightLineIntercept

```
{**** Below  While loop by ASE 15FEB96 it performs a search to see if the
detection time is already in the MasterEntryTime array. If it is, then you do
not want to recalculate the detection time if not needed due to changes in
the scenario. ****}

ThisRec := ASK MasterEntryTime First;
WHILE (ThisRec <> NILREC)
    IF (ThisRec.Plat = ASK Platform ID) AND (ThisRec.Threat = ASK Target ID)
        { Means we have a match }
        IF ThisRec.Sensor = Sensorname
            RandTime := ThisRec.Time;
            PrevAdded := TRUE;
            OUTPUT("Have a match");
        END IF;
        ThisRec := ASK MasterEntryTime Next(ThisRec);
    ELSE
        OUTPUT(" *** Checking MasterEntryTime Array No match");
        ThisRec := ASK MasterEntryTime Next(ThisRec);
    END IF;
 END WHILE;
CASE Solutions
    WHEN 0: t1 := -1000.0;
           t2 := -1000.0;
           EntryExit.Entry.Time := -1000.0;
           EntryExit.Exit.Time := -1000.0;

    WHEN 1: t1 := ABS((TgtLWPoint.Location.x - Sol1) / DeltaVel) + SimTime;
           t2 := t1;
           ASK Environment TO SetEntryLocation(ASK Target  PredictedPosition(t1));
           sensorrec :=  ASK MasterSensorList TO GiveSensor(Sensorname);
           IF (sensorrec.TypeSensor) = "RADAR"
              OUTPUT("Detected a RADAR and executing random routine");
              ASK Environment TO SetCurrentPlats(Platform,Target);
              IF PrevAdded = FALSE {Don't want another rand time}
                 ASK Environment TO FindDetectTime(sensorrec.Attributes.PulseInt, ASK Platform ID,
                                                  sensorrec.SensorTypeName,
                                                                 Target, DetectionRadius);
            END IF;
            DoRandom := TRUE;
          END IF;
         {If random routine is executed do this}
          IF DoRandom
            IF PrevAdded = FALSE
            EntryExit.Entry.Time := t1 + Environment.RandTime;
            DISPOSE(EntryExit.Entry.Location);
            OUTPUT("NEG T1 found in Detect-straightline motion");
            EntryExit.Entry.Location := ASK Target PredictedPosition(t1 +Environment.RandTime);
            EntryExit.Exit.Time := t2 ;
            DISPOSE(EntryExit.Exit.Location);
            EntryExit.Exit.Location := ASK Target PredictedPosition(t1 +Environment.RandTime);
            OUTPUT(" Random additional time to intercept in IDetect: ",Environment.RandTime);
          ELSE
            EntryExit.Entry.Time := t1 + RandTime;
            DISPOSE(EntryExit.Entry.Location);
            OUTPUT("NEG T1 found in Detect-straightline motion");
            EntryExit.Entry.Location := ASK Target PredictedPosition(t1+ RandTime);
            EntryExit.Exit.Time := t2 ;
            DISPOSE(EntryExit.Exit.Location);
            EntryExit.Exit.Location := ASK Target PredictedPosition(t1 +RandTime);
           OUTPUT(" Random time ALREADY IN ARRAY in IDetect: ",RandTime);
         END IF;
      ELSE
         EntryExit.Entry.Time := t1;
         DISPOSE(EntryExit.Entry.Location);
         OUTPUT("NEG T1 found in Detect-straightline motion");
         EntryExit.Entry.Location := ASK Target PredictedPosition(t1);
         EntryExit.Exit.Time := t2 ;
         DISPOSE(EntryExit.Exit.Location);
         EntryExit.Exit.Location := ASK Target PredictedPosition(t1);
      END IF;

    WHEN 2:
       IF (ABS(Distance(TgtLWPoint.Location,PltLWPoint.Location)) -
          DetectionRadius < 0.0 )
          t1 := SimTime();
          IF DeltaVel > 0.0
             t2 := ABS((TgtLWPoint.Location.x - MAXOF(Sol1,Sol2)) /DeltaVel) + SimTime();
          ELSE
             t2 := ABS((TgtLWPoint.Location.x - MINOF(Sol1,Sol2)) /DeltaVel) + SimTime();
          END IF;
       ELSE
          IF DeltaVel > 0.0
             t1 := ABS((TgtLWPoint.Location.x -MINOF(Sol1,Sol2))/DeltaVel) + SimTime();
             t2 := ABS((TgtLWPoint.Location.x -MAXOF(Sol1,Sol2))/DeltaVel) + SimTime();
          ELSE
             t1 := ABS((TgtLWPoint.Location.x -MAXOF(Sol1,Sol2))/DeltaVel) + SimTime();
             t2 := ABS((TgtLWPoint.Location.x -MINOF(Sol1,Sol2))/DeltaVel) + SimTime();
        END IF;
        IF t2 < t1
```

```
            t2 := t1;
          END IF;
        END IF;
        ASK Environment TO SetEntryLocation(ASK Target PredictedPosition(t1));
        sensorrec := ASK MasterSensorList TO GiveSensor(Sensorname);
        IF (sensorrec.TypeSensor) = "RADAR"
         OUTPUT("Detected a ", sensorrec.TypeSensor, " which should match RADAR and
                                                         executing random routine");
         ASK Environment TO SetCurrentPlats(Platform,Target);
         IF PrevAdded = FALSE {Don't want another rand time}
             ASK Environment TO FindDetectTime(sensorrec.Attributes.PulseInt, ASK Platform ID,
                                               sensorrec.SensorTypeName,
                                               Target, DetectionRadius);
         END IF;
         DoRandom := TRUE;
      END IF;
      {If random routine is executed do this}
      IF DoRandom
         IF PrevAdded = FALSE
            EntryExit.Entry.Time := t1 + Environment.RandTime;
            DISPOSE(EntryExit.Entry.Location);
            EntryExit.Entry.Location := ASK Target PredictedPosition(t1 +Environment.RandTime);
            DISPOSE(EntryExit.Exit.Location);
            EntryExit.Exit.Time := t2;
            EntryExit.Exit.Location := ASK Target PredictedPosition(t2);
            OUTPUT(" Random additional time to intercept in IDetect: ",Environment.RandTime);
        ELSE
            EntryExit.Entry.Time := t1 + RandTime;
            DISPOSE(EntryExit.Entry.Location);
            OUTPUT("NEG T1 found in Detect-straightline motion");
            EntryExit.Entry.Location := ASK Target PredictedPosition(t1+ RandTime);
            EntryExit.Exit.Time := t2 ;
            DISPOSE(EntryExit.Exit.Location);
            EntryExit.Exit.Location := ASK Target PredictedPosition(t1 +RandTime);
            OUTPUT(" Random time ALREADY IN ARRAY in IDetect: ",RandTime);
        END IF;
     ELSE
        EntryExit.Entry.Time := t1;
        DISPOSE(EntryExit.Entry.Location);
        EntryExit.Entry.Location := ASK Target PredictedPosition(t1);
        DISPOSE(EntryExit.Exit.Location);
        EntryExit.Exit.Time := t2;
        EntryExit.Exit.Location := ASK Target PredictedPosition(t2);
     END IF;
     OTHERWISE
     WriteLine("Funny return from Quadratic");
  END CASE;
  OUTPUT(" T1 intercept: ",t1);
  OUTPUT(" The time ",ASK Platform ID," will detect ",ASK Target ID, " = ",EntryExit.Entry.Time);
  {Want to take a data point of the random time until detection. This
   stat could also be gathered on only the random times generated earlier.
   Gathering stat here combines any changes in the initial time Ro is
   reached ...}
   StatTime := EntryExit.Entry.Time;
```

```
    ASK StatKeeper TO GetSample(StatTime);
    OUTPUT("Gathered a Stat");
END METHOD;
```

Many other changes were made to the NPS Platform Foundation that will not be discussed in this appendix. The changes insured the flow of the correct information to the proper modules. This information exchange, although initially done only for this thesis, allows for many other uses of the NPS Platform Foundation. All new and modified modules for this thesis are listed below. An asterisk denotes new modules.

- *D/IAARON1.mod* *
- *D/IAARONCDF.mod* *
- *D/IAARONTIME.mod* *
- *D/ICPA.mod*
- *D/IDetect.mod*
- *D/IDetRng.mod*
- *D/IMathFunc.mod* *
- *D/IPList.mod*
- *D/IReact.mod*
- *D/ISpace.mod*
- *D/IVS.mod*
- *D/IVSMan.mod*
- *D/IWrite.mod*
- *D/IGP.mod*
- *IGPMast.mod*

- *IGSimExc.mod*
- *IManuv.mod*
- *IRep.mod*

LIST OF REFERENCES

1. Law, A. M., Kelton, W. D., *Simulation Modeling and Analysis*, McGraw-Hill, Inc., New York, NY, 1991.

2. Hovanessian, S. A., *Introduction to Sensor Systems*, Artech House, Inc., Norwood, MA, 1988.

3. Hardenburg, W., Facsimile Notes, Naval Research Laboratory, August 9, 1995.

4. Meyer, D. A., Mayer, H. A., *Radar Target Detection, Handbook of Theory and Practice*, Academic Press, New York, NY, 1973.

5. Marcum, J. I., "A Statistical Theory of Target Detection," *Detection and Estimation Applications to Radar*, Reedited by Haykin, S. S., Dowden, Hutchinson & Ross, Inc., Stroudsburg, PA, 1976.

6. Blake, L. V., *Radar Range-Performance Analysis*, D.C. Heath and Company, Lexington, MA, 1980.

7. Hovanessian, S. A., *Radar System Design and Analysis*, Artech House, Inc., Dedham, MA, 1984.

8. Swerling, P., "Probability of Detection for Fluctuating Targets," *Detection and Estimation Applications to Radar*, Reedited by Haykin, S. S., Dowden, Hutchinson & Ross, Inc., Stroudsburg, PA, 1976.

9. Bailey, M., *The Naval Postgraduate School Platform Foundation*, Technical Report NPS-OR-94-005, Department of Operations Research, Naval Postgraduate School, Monterey, CA, 1994.

10. *Jane's Radar and Electronic Warfare Systems*, Seventh Edition, Jane's Information Group Inc., Alexandria, VA, 1995.

11. *MODSIM II Reference Manual*, CACI Products Co., La Jolla, CA, 1994.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, CA 93943-5101	2
3.	Chairman, Code OR Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Professor Arnold Buss, Code OR/Bu Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5121	4
5.	Professor Jams Eagle, Code OR/Er Department of Operations Research Naval Postgraduate School Monterey, CA 93943-5121	4
6.	Dr. William Hardenburg, Code 5740 Naval Research Laboratory 4555 Overlook Ave, SW Washington, D.C. 22193	4